Root Cause Analysis with Enriched Process Logs

S. Suriadi[1], C. Ouyang[1], W.M.P. van der Aalst[1,2], and A.H.M. ter Hofstede[1,2]

[1] Queensland University of Technology, Australia
{s.suriadi,c.ouyang,a.terhofstede}@qut.edu.au
[2] Eindhoven University of Technology, The Netherlands
{w.m.p.v.d.aalst,a.h.m.ter.hofstede}@tue.nl

Abstract. In the field of process mining, the exploitation of event logs for the purpose of root cause analysis is a relatively unexplored area. This paper proposes an approach to enrich and transform process-based logs for the purpose of root cause analysis through the application of *classification* techniques. This approach facilitites an *objective* identification of the root cause(s) of various incidents as it is grounded on what has *actually* happened, thus, minimizing the influence of subjective perceptions in the results. The approach proposed in this paper is formalized and its applicability has been validated using both self-generated logs and a publicly-available log.

Topic classification: Mining of business processes

Keywords: process mining, root cause analysis, event logs, business process management

1 Introduction

A recent survey by Gartner shows that risk management is one of the top strategic business priorities for CEOs and senior executives [2]. A common way to mitigate risks is to remove or minimize the presence of key factors which are known to contribute to the occurrence of unwanted risk events. For example, it is well-established that pilot fatigue is one of the key factors contributing to airline safety issues [3]. Thus, measures to mitigate the chance of pilots being fatigued during working hours have routinely been implemented

Root cause analysis (RCA) has been applied widely in organisations and many techniques have been developed, including the use of flow charts, spider charts, brainstorming, and others [1]. Through RCA, one aims to find an explanation of risk incidents, preferably an explanation involving a minimal number of factors. As many events are recorded in logs nowadays (e.g. by devices or software applications) one can exploit this data for the purpose of RCA with the added advantage that the data reflects reality and not a perception (e.g. in the form of a model) of reality. As we focus on operational rocesses in this paper, one can take post-execution event data as a starting point and compare features, recorded as attributes, of cases (i.e. process instances) that were classified as successful with those features of cases that were classified as unsuccessful.

This way we can define a *classification* problem whose results provide valuable input for RCA.

The features which may be the root cause(s) of risk incidents may come from various context, including instance context, process context, social context, and environmental context [19]. However, raw data from post-execution logs (commonly known as event logs), may not readily contain the necessary information to explain the possible root cause(s) of risk incidents.

In this paper, we propose an RCA approach that starts with an existing event log and enriches it with relevant contextual information for explaining possible root cause(s) of risk incidents. This enriched log can then be manipulated further such that it can be analyzed using established *classification* techniques. In other words, we transform a process mining problem, i.e. RCA based on event logs, into a standard *classification* problem. In this paper, we focus on the approach and not on the selection of variables (features). In fact, in order to illustrate our approach and also to scope our paper, we investigate the effect of workload on 'overtime' (long-running) cases. Here we may draw on the *Yerkes Dodson* law [21, pp. 485-486] which captures the relationship between the stress level of resources and their performance. This relationship can be visualised as an inverted letter 'U' in a diagram with performance on the y-axis and degree of stress level on the x-axis, indicating that both low and high stress levels are related to low performance.

In the remainder of the paper, we first explain our approach in detail and present a formalisation. This is then followed by an application of the approach to RCA with workload-enriched logs. Finally the approach is validated with a publicly available log to demonstrate its applicability.

2 Approach

In this section, we describe our approach to mining the root cause of risk incidents (such as budget overrun and long running/overtime cases) through the *transformation* of a process mining problem into a standard *classification* problem. Our approach starts with determining relevant information that is needed to explain the root cause of a risk incident, followed by the enrichment of the related event log with the necessary information to ensure that sufficient information for RCA is captured. Through the application of aggregation functions and other refinement procedures, we transform this enriched event log into a form that is suitable to be analysed by *classification* techniques. Fig. 1 depicts our approach - the details of which are explained in this section.

RCA Data Requirements. As a requirement from *classification* techniques, RCA uses a *response* variable for the labelling of 'successful/unsuccessful' cases. An example of a response variable can be seen as a feature in a log which states the outcome of a case as either being 'on-time' or 'overtime'. The features which may influence the occurrence of 'successful/unsuccessful' cases are labelled as *predic-*

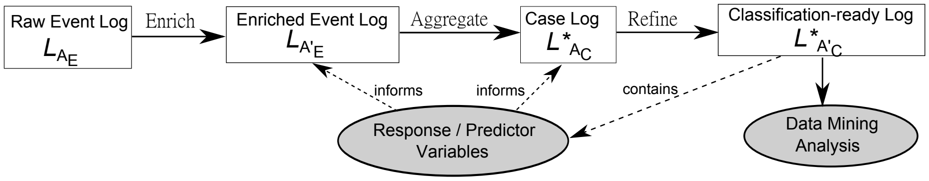

Fig. 1: Approach Diagram.

tor variables. An example of a predictor variable can be specified by workload which may have impact on the occurrence of long-running (over-time) cases.

Thus, the very first step of our approach is to determine the *response* variable capturing the risk incident we want to analyze, and the set of *predictor* variables which may contribute to the occurrence of such risk incident. The determination of these attributes must of course be informed by established management theories and studies as well as from a company's risk registries. In process mining, these variables may be taken from various context, including instance context (e.g. size of an insurance claim), process context (e.g. total number of resources involved), social context (e.g. stress level), and external context (e.g. weather) [19].

Enrich Raw Event Log. Once the *predictor* and *response* variables are determined, we need to verify if our raw event log contains sufficient information to captures those variables. It is possible that some information may not be readily available in the raw event log, but can be derived from the data in the raw event log. There are also situations when the required information needs to be obtained from external sources and be correlated to the raw event log. Once the above information is obtainable, we can then extend the raw event log with such information thus resulting in an enriched event log).

Definition 1 (Event log). An *event log* consists of a set of events. Events are characterised by various attributes, e.g., an event may have a timestamp, correspond to an activity, etc. Let $A_E = \{a_1, ..., a_n\}$ be a set of event attributes. For each attribute $a_i \in A_E$ $(1 \leqslant i \leqslant n)$, $D_i = dom(a_i)$ (i.e. the domain of a_i), e.g., the domain of attribute *timestamp* is the set of all possible time values. $L_{A_E} \subseteq 2^{D_1 \times \ldots \times D_n}$ is an *event log* based on the set of attributes A_E. For any event $e \in L$ and attribute $a \in A_E$: $\#_a(e)$ is the value of attribute a for event e.

Definition 2 (Event log enrichment). Let L_{A_E} be an event log based on the set of attributes A_E. Let A_D be a set of derivable attributes whose value can be calculated from A_E, i.e. for each attribute $d \in A_D$, $L_{A_E \cup \{d\}} = \delta_d(L_{A_E})$ where δ_d is a derivative function that computes the value of attribute d for all events in L_{A_E}. Let A_K be a set of correlatable attributes whose value can be obtained from external sources based on some key attributes in A_E, i.e. for each attribute $k \in A_E$, $L_{A_E \cup \{k\}} = \omega_k(L_{A_E})$ where ω_k is a correlative function that retrieves the value of attribute k for all events in L_{A_E}. $L_{A'_E}$ where $A'_E = A_E \cup A_D \cup A_K$, is an *enriched* event log with a set of extended (derivable and correlatable) attributes.

From Event Log to Case Log. The next step is to transform the event log into a case log. This is necessary because RCA is normally performed at case level.

Definition 3 (Case log). A *case log* comprises a set of cases. Cases, like events, have attributes, e.g., a case has a case identifier, corresponds to a trace (i.e. a sequence of events), etc. Let $A_C = \{s_1, ..., s_m\}$ be a set of case attributes, including a mandatory attribute named *case_id* which uniquely identifies a case. For each $s_i \in A_C$ $(1 \leqslant i \leqslant m)$, $D_i = \mathit{dom}(s_i)$ where $\mathit{dom}(\mathit{case\_id}) = \mathcal{C}$ (i.e. the set of all possible case identifiers). $L^*_{A_C} \subseteq 2^{D_1 \times \ldots \mathcal{C} \times \ldots D_m}$ is a case log based on the set of attributes A_C. For any case $c \in L^*_{A_C}$ and attribute $s \in A_C$: $\#_s(c)$ is the value of attribute s for case c. For any $c_1, c_2 \in L^*_{A_C}$, $\#_{\mathit{case\_id}}(c_1) \neq \#_{\mathit{case\_id}}(c_2)$.

Definition 4 (Aggregation of event log to case log). Let L_{A_E} be an event log based on the set of event attributes A_E where $\mathit{case\_id} \in A_E$. A corresponding case log $L^*_{A_C}$ based on the set of attributes A_C can be generated from L_{A_E} as: (1) the set of case identifiers $\bigcup_{c \in L^*_{A_C}} \#_{\mathit{case\_id}}(c) = \bigcup_{e \in L_{A_E}} \#_{\mathit{case\_id}}(e)$; and (2) for each case $c \in L^*_{A_C}$ and attribute $s \in A_C \backslash \{\mathit{case\_id}\}$, $\#_s(c) = \alpha_s(L_{A_E}, \#_{\mathit{case\_id}}(c))$ where α_s is an *aggregation function* that calculates the value of attribute s in case c from the relevant events in L_{A_E}.

Get Case Log Ready for Classification. For *classification* techniques to work, there are often requirements imposed on the structure of the data. For example, most *classification* techniques require the *response* variable to be presented in a categorial (nominal) form. If our data is not already in the correct format, we need to further transform the attributes in a case log accordingly.

Definition 5 (Classification-ready case log). Let $L^*_{A_C}$ be a case log based on the set of (normal) attributes A_C. Let $A^\diamond_C$ be a set of classification attributes (e.g. capturing response variables) whose value is of categorial form and can be computed from the set of case attributes A_C. $L^*_{A'_C}$ where $A'_C = A_C \cup A^\diamond_C$, is a *classification-ready case log*. For each case $c \in L^*_{A'_C}$ and attribute $q \in A^\diamond_C$: $\#_q(c) = \chi_q(L^*_{A_C}, c)$ where χ_q is a classification function that derives the value of attribute q in case c based on the corresponding case data in $L^*_{A_C}$.

RCA with Classification Techniques. Through the process detailed so far, we have now transformed a raw event log into a case log in a format that is suitable for classification analysis. The enriched case log now has the values of both the *response* and *predictor* variables of interest. One can feed these data into various data mining algorithms to find the root cause(s) of a risk incident.

3 RCA with Workload-Enriched Logs

In this section, we apply our approach to the analysis of *overtime cases* using event logs enriched with *workload* information. According to dictionary, workload is defined as "the amount of work assigned to or expected from a person in a specified time period"[1]. In the context of business processes, workload has been defined and computed in different ways (e.g. [21, Chapter 11] [12]). In this paper, we consider *workload as a list of work items that have been allocated to and/or started by a resource in a specified time period.*

Workload is usually not recorded directly in a raw event log but can be derived from a set of basic event attributes that capture the information about resource, activity, timestamp, and transaction type (which refers to the life-cycle of activities, e.g. start, complete) associated with each event. Below, we precisely define workload based on the information in a given event log of any process.

Definition 6 (Workload). Let L be an event log based on the set of event attributes including $\{case\_id, resource, activity, time, trans\}$, where $dom(trans) = \{allocate, start, complete\}$. For any resource $r \in \{\#_{resource}(e) \mid e \in L\}$, the workload of r in time period (t_s, t_e), is defined as $WL_L^{t_s,t_e}(r) = WL_L^{t_s}(r) \cup WL_L^{(t_s,t_e)}(r)$.

- first of all, assume three short notations:
  - $L^{r,t_s} = \{e \in L \mid \#_{resource}(e) = r \wedge \#_{time}(e) < t_s\}$
  - $L^{r,t_s,t_e} = \{e \in L \mid \#_{resource}(e) = r \wedge t_s \leqslant \#_{time}(e) \leqslant t_e\}$
  - $\forall e \in L, wi(e) = (\#_{activity}(e), \#_{case\_id}(e))$. Then:
- $WL_L^{t_s}(r) = \{wi(e) | e \in L^{r,t_s} \wedge \#_{trans}(e) \in \{allocate, start\}\} \setminus \{wi(e) | e \in L^{r,t_s} \wedge \#_{trans}(e) = complete\}$, i.e. the list of work items that have been allocated to and/or started but not yet completed by resource r at time t_s; and
- $WL_L^{(t_s,t_e)}(r) = \{wi(e) \mid e \in L^{r,t_s,t_e} \wedge \#_{trans}(e) \in \{allocate, start\}\} \setminus WL_L^{t_s}(r)$, i.e. the list of *additional* work items that are allocated to and/or started by resource r within time period (t_s, t_e).

Consider a raw event log L_{A_E} with a set of existing event attributes A_E. Following the approach in Sect. 2, we enrich log L_{A_E} with workload information. A derivable attribute *res_workload* captures the workload of all resources in the log within certain time periods. Each time period is specified by referencing the timestamp of an individual event in L_{A_E} and using the *average activity duration* (AAD) of the process as the time duration. We assume an existing function *AvgActDuration* which takes an event log and returns the corresponding AAD[2].

Algorithm 1 defines how to compute the value of attribute *res_workload* based on the raw event log L_{A_E}. First, a relation R_{WL}, i.e. a set of (*time*,*res_workload*) tuples, is derived capturing the workload of all resources each time an event is recorded in L_{A_E}. By definition, an event log can be treated as a relation where the set of event attributes specify its relation scheme. Hence, in Algorithm 1 three relational algebra operators [10] are used to define certain log operations: the *selection* (σ) of a set of events that satifies specific conditions in the log; the *projection* (π) of all events in the log restricted to a set of attributes; and finally the *natural join* ($\bowtie$) for combining all events in L_{A_E} and all tuples in R_{WL} on their common attribute *time* to yield the workload-enriched log $L_{A_E \cup \{res\_workload\}}$.

[1] `http://www.thefreedictionary.com/workload`

[2] The underlying computation for such function is already supported by a number of log pre-processing tools, such as Nitro (see `http://www.fluxicon.com/nitro/`).

Algorithm 1: Generation of workload-enriched event log based on an instantiation of derivative function γ in Definition 2.

```
Input: an event log $L_{A_E}$ and a derivable attribute $res\_workload$
Output: a workload-enriched event log $L_{A_E \cup \{res\_workload\}}$
begin
    $R_{WL} := \emptyset$; /* $R_{WL}$ is a relation of $\{time, res\_workload\}$ */
    $R := \{\#_{resource}(e) \mid e \in L_{A_E}\}$;
    $T := \{\#_{time}(e) \mid e \in L_{A_E}\}$;
    $d_h := AvgActDuration(L_{A_E})/2$;
    for $t \in T$ do
        $rwl := \emptyset$;
        $t_s := t - d_h$;
        $t_e := t + d_h$;
        for $r \in R$ do
            $L^r := \sigma_{resource=r}(L_{A_E})$;
            /* calculate allocated/started but not completed work items at time $t_s$ */
            $L^{r,t_s} := \sigma_{time<t_s}(L^r)$;
            for $tr \in \{allocate, start, complete\}$ do
                $L^{tr}_{WI} := \pi_{\{case\_id, activity\}}(\sigma_{trans=tr}(L^{r,t_s}))$
            $rwl := rwl \cup \{(r, (L^{allocate}_{WI} \cup L^{start}_{WI}) \setminus L^{complete}_{WI})\}$;
            /* calculate allocated/started work items within time period $(t_s, t_e)$ */
            $L^{r,t_s,t_e}_{M} := \sigma_{t_s \leqslant time \leqslant t_e}(L^r)$;
            for $tr \in \{allocate, start\}$ do
                $L^{tr}_{WI} := \pi_{\{case\_id, activity\}}(\sigma_{trans=tr}(L^{r,t_s,t_e}))$
            $rwl := rwl \cup \{(r, (L^{allocate}_{WI} \cup L^{start}_{WI}))\}$
        $R_{WL} := R_{WL} \cup \{(t, rwl)\}$
    $L_{A_E \cup \{res\_workload\}} = \bowtie_{\{time\}} (L_{A_E}, R_{WL})$
```

Now consider the set of event attributes $A_C = \{case\_id, time\_start, time\_end, avg\_workload, resources\}$ that are required for deriving the value of predictor and response variables in the next step. We generate case log $L^*_{A_C}$ from (the above enriched) event log $L_{A_E \cup \{res\_workload\}}$. For each case $c \in L^*_{A_C}$, we calculate:

- $\#_{time\_start}(c) = Min(\pi_{\{time\}}(\sigma_{case\_id=\#_{case\_id}(c)}(L_{A_E})))$
- $\#_{time\_end}(c) = Max(\pi_{\{time\}}(\sigma_{case\_id=\#_{case\_id}(c)}(L_{A_E})))$
- $\#_{resources}(c) = \{\#_{resource}(e) \mid e \in \sigma_{case\_id=\#_{case\_id}(c)}(L_{A_E})\}$
- $\#_{avg\_workload}(c) = \{(r, (\sum_{e_i \in L^c_{A_E}} |workload(r, e_i)|\ )\ /\ |L^c_{A_E}|) \mid r \in R\}$ where
  - $R = \{\#_{resource}(e) \mid e \in L_{A_E}\}$ is the set of all resources in L_{A_E},
  - $L^c_{A_E} = \sigma_{\#_{time\_start}(c) \leqslant timestamp \leqslant \#_{time\_end}(c)}(L_{A_E})$ is the set of all events in L_{A_E} that occurred between the start and the end of case c,
  - $workload(r, e_i)$ is the workload of resource r at the time of event e_i, which is specified in $\#_{res\_workload}(e_i)$.

Finally, two examples of classification-ready attributes are $A^{\diamond}_C = \{isOvertime, isClerkInvolved\}$, where *isOvertime* signals if a case (time) duration exceeds a specific *threshold* value, and *isClerkInvolved* indicates if resource named *Clerk* is involved in a case. Both returns a value of boolean type (a valid categorial form). Hence, for each case $c \in L^*_{A'_C}$ where $A'_C = A_C \cup A^{\diamond}_C$,

- $\#_{isOvertime}(c) = ((\#_{time\_end}(c) - \#_{time\_start}(c) > \mathit{threshold})$
- $\#_{isClerkInvolved}(c) = (Clerk \in \#_{resources}(c))$

4 Validation of Approach

Two rounds of approach validation were conducted. The first validation round was based on a self-generated synthetic log. It was conducted during the development of our approach (w.r.t the enrichment of event log with workload information) to facilitate step-by-step validation and refinement of our approach. The second evaluation round was conducted to demonstrate the *general applicability* of our approach in a less controlled environment by applying our approach to a log whose generation was beyond our control. The first validation round is briefly described in Sect. 4.1, while a more elaborate description of the second evaluation round is provided in Sect. 4.2.

Log Quality. To derive workload information, we assume the existence of an event log of certain quality: (1) a minimum of 3-star log quality (as defined in the process mining manifesto [18]), (2) the existence of a number of open XES-equivalent attributes: (a) `concept:name` (at trace and event level), (b) `lifecycle:transition`, `org:resource`, `time:timestamp` (at event level), and (3) the recording of `start` (or `assign`) and `complete` transition of each activity.

4.1 Validation with Synthetic Log

The first validation round was conducted using a Purchase Order scenario. A synthetic log was generated using a Coloured Petri Net (CPN) [9] model of the Purchase Order process through the application of the MXML-log-generator plug-in [11]. This log records the `assign`, `start`, and `complete` transitions for each activity. This log file was imported to a `MySQL` database such that relevant relational algebra operations can be performed. The workload derivation function and case log aggregation functions (as defined in Sect. 3) were implemented as a `Java` program which interacted with the database. The final *classification-ready* case log was successfully fed into the `WEKA` data mining tool such that the root cause of 'overtime' cases can be analyzed by the application of various *classification* algorithms (e.g. `J48` [13] and `JRip` [4]). The *classification* analysis result from this log showed that average resources' workload was a key factor in the occurence of 'overtime' cases.

This validation round confirms the applicability of our approach in performing an RCA through the enrichment and transformation of a process log such that root cause(s) of a risk incident can be identified by applying *classification* techniques. The details of this validation round are available in Appendix A.

4.2 Validation with Public Log

While the first validation round was necessary to confirm the applicability of our approach, it was weak as it was based on a log that we generated ourselves. To demonstrate the general applicability of our approach, we conducted a second validation round using an event log that was generated by other people and is available from the process mining website.[3] Equally important, while further refinement is still needed, the second validation round confirms the generalizability of the `Java` program developed in enriching and transforming event log with resources' workload and involvement information. Thus, it is *possible to automate and streamline* (to a certain degree) the approach proposed in this paper.

The log used in this validation round was generated from a telephone repair process. This log contains the necessary attributes as defined in M. There are four activities in this log: `'Analyze Defects'`, `'Repair (Complex)'`, `'Repair (Simple)'`, and `'Test Repair'`. There are 12 resources in the log: `SolverC1, SolverC2, SolverC3, SolverS1, SolverS2, SolverS3, Tester1, Tester2, ..., Tester6`. There are more than 1100 cases (mostly completed cases).

Log Enrichment with Workload The log used in this validation round meets the minimum log quality as stated earlier. Thus, we can proceed to enrich the log with workload information. This log was imported to a `MySQL` database table (called `eventlog` table) so that it could be manipulated using relational algebra operators (see Table 1 for a snippet of the table).

`caseID`	`Activity`	`eventType`	`timestamp`	`resource`
18	Analyze Defect	start	1970-01-01 15:36:00	Tester6
18	Analyze Defect	complete	1970-01-01 15:44:00	Tester6
15	Repair (Complex)	start	1970-01-01 19:29:00	SolverC3
....	...	...	...	...

Table 1: A snippet of the `eventlog` database table.

The workload derivation function (Algorithm 1 from Sect. 3) was implemented in a `Java` program (which interacted with the `eventlog` through SQL queries). This program outputs a table called `workloadstarted` which captures the workload information of each resource (i.e. the realization of R_{WL} relation). The *average activity duration* used is 450 seconds. The log used in our validation only contains the `start` and `complete` transition, thus, `workloadstarted` only reflects the work items that were still being executed by a particular resource at a point in time when an event was recorded in the log. A snippet of the `workloadstarted` table is provided in Table 2. The content of each field named after each resource identifier (e.g. `SolverC1` and `SolverC3`) represents the workload of the corresponding resource at its corresponding timestamp. For example, using `1970-01-03 00:37:00` as a reference time, resource `SolverC1` had no work items being executed (or assigned to) between the period of `1970-01-03 00:33:15` and `1970-01-03 00:40:45`, while resource `SolverS3` had two work items that were either being executed or assigned (that is, the activity `'Repair`

[3] `http://www.processmining.org/_media/tutorial/repairexample.zip`

`(Simple)'` for case number 70 and 34) over the same period. The *workload-enriched* event log (i.e. $L_{A'_E}$) is obtained by *joining* Table 2 and Table 1 *on* `timestamp` field.

`timestamp`	`Workload`$_{SolverC1}$	...	`Workload`$_{SolverC3}$	`Workload`$_{SolverS3}$	...
1970-01-03 00:37:00	(empty)	...	(empty)	(70:Repair (Simple)), (34:Repair (Simple))	... ...
...	...	...	...	...	...
1970-01-03 14:10:00	(36:Repair (Complex))	...	(60:Repair (Complex))	(empty)	...
...	...	...	...	...	...

Table 2: A snippet of the `workloadstarted` table (R_{WL})

Aggregation to Case Log The *workload-enriched* event log obtained so far contains the workload information of each resource at every timestamp in the log. Given that a case may consist of more than one event, we need to aggregate the workload information among all relevant events (that make up one case) in the log. We have implemented the necessary functions to obtain the $\#_{time\_start}(c)$, $\#_{time\_end}(c)$, $\#_{resources}(c)$, and $\#_{avg\_workload}(c)$ case attributes (defined in Sect. 3). In our implementation, the average workload for each resource (quantified as the number of work items) is stored in a separate field, each named using the following format: `avgWL`$_{r_x}$, where r_x is a resource's identifier (see Table 3). The results of the application of these aggregation functions were stored in another table, called the `caseLog` table (which is a manifestation of $L^*_{A_C}$).

Classification-ready Case Log Finally, we transformed `caseLog` into a form that is ready for classification analysis. Currently, `caseLog` does not contain a proper *response* variable: we know the start and end of each case; however, we still need to further refine this information to obtain the duration of the case and then to categorize each case to 'on-time' or 'overtime'. In other words, we need to implement a function to obtain the $\#_{isOvertime(c)}$ attribute (defined in Sect. 3). In our implementation, 'on-time' cases were labeled with a `'0'` value, while 'overtime' cases were labelled with a `'1'` value. The `threshold` value which determines if a case is overdue or not is set to 1 hour.[4]

We have also implemented a function to obtain the $\#_{isRxInvolved(c)}$ value (defined in Sect. 3) to ensure that the information regarding the involvement (or non-involvement) of each resource is presented as a distinct *predictor* variable. For n-number of resources, this function creates n new field, named as `isInvolved`$_{r_x}$ (where r_x is a resource's identifier). Each of these fields can now be labelled individually as a distinct *predictor* variable.

We then extended the `caseLog` table with the results of the application of these two functions. A snippet of the final *classification-ready* case log table is shown in Table 3. This table is a manifestation of $L^*_{A'_C}$

[4] This number is chosen based on the fact that most cases completed within 1 hour (of course, the assumption here is that the majority of cases do not run overtime).

caseID	start	end	isOverdue	avgWL$_{SolverC1}$	...	avgWL$_{Tester6}$	isInvolved$_{SolverC1}$	...
1070	1970-01-23 20:58:00	...	1	1.36364	...	0	false	...
1075	1970-01-24 13:22:00	...	0	0.782609	...	1.6087	false	...

Table 3: A snippet of a *classifier-ready* case log ($L^*_{A'_C}$).

RCA with Classification Techniques We fed the *classification-ready* case log into `WEKA` [23]. `WEKA` supports *classification* algorithms, such as `J.48` [13], `JRip` [4], and many others. We started the analysis by just using the average workload of all resources as *predictor* variables. An example of the results of the decision tree learning analysis (using the `J.48` algorithm [13] under 10-fold cross validation mode [17, Chapter 3]) is shown in Fig. 2 (top part). For example, line 3 in Fig. 2 shows that if the average workload of resource `SolverS1` is equal or less than 0.117647 and the average workload for `Tester4` is equal to or less than 0.909091 (line 5), then the rules generated by the algorithm say that all 248 cases which fulfill these criteria should be classified as 'on-time' (represented as `'0'`). While the records in the log show that this rule misclassified 42 cases, this rule is correct in the majority of cases (202 cases out of 248).

However, the accuracy of the classification result is rather low. For example, for 'on-time' cases, the true positive (TP) rate (that is, the proportion of all 'on-time' cases which were correctly classified as 'on-time') is ≈ 70% ; however, the false positive (FP) rate (that is, the proportion of all 'overtime' cases which were incorrectly classified as 'on-time') is as high as 51%. For 'overtime' cases, the TP rate (that is, the percentage of 'overtime' cases which were correctly classified as 'overtime') is ≈ 48%, while the FP rate is as high as 29%. Fig. 3 shows the full accuracy metrics. This analysis result suggests that the average workload for a number of resources seem to have an influence the occurrence of long-running cases, however, the correlation seems rather weak.

Beyond Workload - Adding Another Predictor Variable We decided to include another predictor variable in our analysis, namely the resource involvement information (recall that this information was already calculated in our *classification-ready* case log). The same `J.48` algorithm was executed, except that this time, we used both workload and resource involvement information as *predictor* variables. The result obtained was much more accurate: 92% TP rate/ 14% FP rate (for 'on-time' cases), and 85% TP rate/7% FP rate (for overtime cases). Both predictor variables were used in the generated classification tree; however, the complexity of the generated classification tree is high - this may suggest an overfitting model (the decision tree is not shown in this paper).

We also applied a rule-based classification algorithm to the same data set, namely, the `JRip` algorithm. The result obtained was much simpler than the one obtained from `J.48` algorithm (only 4 rules - see Fig. 2 bottom part). For example, the rule in line 28 states that all 201 cases in which `SolverC3` is involved will run 'overtime' (in reality, this rule only misclassified 6 cases). More interestingly, however, is that the generated rules do not include resources' workload as a factor in lovertime cases (although they were included in the analysis). The accuracy of the result was also comparable: ≈ 92%/14% (TP/FP rate) for 'on-

```
J.48 Classification Tree - Predictor Variables: only average workload for all resources
=====================================================================================
avgWL_SolverS1 <= 0.117647
|   avgWL_Tester4 <= 0.981818
|   |   avgWL_Tester4 <= 0.909091: 0 (248.0/42.0)
|   |   avgWL_Tester4 > 0.909091: 1 (67.0/24.0)
|   avgWL_Tester4 > 0.981818
|   |   avgWL_Tester4 <= 1.05882: 0 (204.0/50.0)
|   |   avgWL_Tester4 > 1.05882
|   |   |   avgWL_Tester1 <= 0.909091: 1 (6.0)
|   |   |   avgWL_Tester1 > 0.909091
|   |   |   |   avgWL_Tester2 <= 0.914286: 1 (3.0)
|   |   |   |   avgWL_Tester2 > 0.914286
|   |   |   |   |   avgWL_Tester3 <= 0.9: 0 (8.0)
|   |   |   |   |   avgWL_Tester3 > 0.9
|   |   |   |   |   |   avgWL_SolverC1 <= 0.818182: 0 (3.0)
|   |   |   |   |   |   avgWL_SolverC1 > 0.818182: 1 (4.0)
avgWL_SolverS1 > 0.117647
|   avgWL_Tester4 <= 0.310345
|   |   avgWL_SolverS2 <= 1.08333: 0 (174.0/65.0)
|   |   avgWL_SolverS2 > 1.08333
|   |   |   avgWL_Tester2 <= 0.324324: 0 (2.0)
|   |   |   avgWL_Tester2 > 0.324324: 1 (9.0)
|   avgWL_Tester4 > 0.310345: 1 (376.0/155.0)

JRip Algorithm - Predictor Variables: average workload and resources involvement
================================================================================
(isInvolved_SolverC3 = 1) => isOverdue=1 (201.0/6.0)
(isInvolved_SolverS1 = 1) => isOverdue=1 (206.0/41.0)
(isInvolved_SolverC2 = 1) and (isInvolved_SolverC1 = 1) => isOverdue=1 (22.0/3.0)
 => isOverdue=0 (675.0/64.0)
```

Fig. 2: J.48 Classification Tree - Analysis Results

```
Full Accuracy Metrics
=====================
Correctly Classified Instances         684               61.9565 %
Incorrectly Classified Instances       420               38.0435 %
Kappa statistic                          0.1981
Mean absolute error                      0.4323
Root mean squared error                  0.4932
Relative absolute error                 89.9536 %
Root relative squared error            100.6253 %
Total Number of Instances             1104

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.708     0.512      0.673     0.708     0.69       0.633    0
                 0.488     0.292      0.528     0.488     0.507      0.633    1
Weighted Avg.    0.62      0.424      0.615     0.62      0.617      0.633

=== Confusion Matrix ===

   a   b   <-- classified as
 468 193 |   a = 0
 227 216 |   b = 1

```

Fig. 3: Full Accuracy Metrics for the J.48 Classification Tree (from Fig. 2 - top part)

time' cases, and ≈ 85%/7% (TP/FP rate) for 'overtime' cases. Fig. 4 shows the full accuracy metrics of the JRrip rules.

```
Full Accuracy Metrics
====================
Correctly Classified Instances         989               89.5833 %
Incorrectly Classified Instances       115               10.4167 %
Kappa statistic                          0.782
Mean absolute error                      0.1806
Root mean squared error                  0.3027
Relative absolute error                 37.5884 %
Root relative squared error             61.7519 %
Total Number of Instances             1104

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.924     0.147      0.904     0.924     0.914      0.888    0
                 0.853     0.076      0.883     0.853     0.868      0.888    1
Weighted Avg.    0.896     0.118      0.896     0.896     0.896      0.888

=== Confusion Matrix ===

   a   b   <-- classified as
 611  50 |   a = 0
  65 378 |   b = 1
```

Fig. 4: Full Accuracy Metrics for the JRip Rules (from Fig. 2 - bottom part)

By comparing these results, we may say that both workload and resource involvement may be the root causes of overtime cases in our log; however, the nature of resource involvement alone may already be sufficient to explain the occurrence of 'overtime' cases in most instances.

5 Related Work

Traditional RCA techniques have been frequently applied in the domain of business and risk management (e.g. Andersen and Fagerhaug [1] and Wilson et al. [22]), as well as manufacturing industries (e.g. Horev [8]). Similarly, the use of data mining techniques in business domain has also been heavily studied and applied (e.g. Cao et al. [5]). However, the use of data mining techniques to systematically perform an RCA of business process-related issues is still a relatively unexplored area in the field of process mining (and business process management in general), although a limited number of related studies have started to emerge. For example, Heravizadeh et al [7] proposed a technique to understand the root cause of business processes based on business process models and augmented it with concepts from requirement engineering - this is in contrast with our approach which is based on post-execution data.

Rozinat and van der Aalst [16] proposed the use of *classification* technique to find the correlation between data attributes and the routing choices made in business processes. This approach is different from ours in that our approach focuses on the enrichment and transformation of event logs to classification-ready log for the purpose of RCA. Of course, the classification-ready log can always be used to find the 'root cause' of various routing decisions; however, it is not limited to such cases only.

Furthermore, in the work by Nakatumba and van der Aalst [12], a form of RCA which seeked to study the correlation between resources' workload and their performance were performed using process mining techniques and data mining technique (linear regression). Similarly, in the work by Rozinat et al. [15], the results obtained from a process mining exercise were used to explain the root cause of the occurence of idle times in a manufacturing test process. Nevertheless, the focus of both approaches was on the RCA itself, instead of the *approach* taken to facilitate such an analysis. The approach proposed in this paper attempts to fill this gap.

6 Conclusions

We have presented and formalized an approach to enrich and transform event log into a form that allows an RCA based on *classification* techniques. The applicability of our approach to facilitate RCA of risk incidents has been validated using both self-generated synthetic log and publicly available log. Future work include the validation of our approach using real-life log, as well as the packaging of our approach as a plug-in to the process mining tool (ProM). The application of optimization techniques to select the best set of features as the root cause(s) of a risk incident will also be considered.

References

1. Bjorn Andersen and Tom Fagerhaug. *Root Cause Analysis: Simplified Tools and Techniques*. ASQ Quality Press, 2nd edition, 2006. ISBN-12 978-0-87389-692-4.
2. French Caldwell. CEO survey 2012: CIOs must link risk management and compliance to business priorities. *Gartner*, (G00226165), March 2012.
3. Stephanie Chen. Pilot fatigue is like 'having too much to drink'. *CNN News*, May 2009.
4. William W. Cohen. Fast effective rule induction. In *Proceedings of 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.
5. Longbing Cao et al., editor. *Data Mining for Business Applications*. Springer, 2009.
6. Christian W. Günther and Wil M. P. van der Aalst. A generic import framework for process event logs. In *BPM Workshops on Business Process Intelligence*, volume 4103. LNCS, 2006.
7. Mitra Heravizadeh et al. Root cause analysis in business processes. Technical report, Queensland University of Technology, 2008.
8. Menachem Horev. *Root Cause Analysis in Process-Based Industries*. Trafford Publishing, 2008.
9. Kurt Jensen and Lars M. Kristensen. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. Springer, 2009.
10. David Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

11. Ana K.A. De Medeiros and Christian W. Günther. Process mining: Using CPN tools to create test logs for mining algorithms. In *6th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 177–190, 2005.
12. Joyce Nakatumba and Wil M. P. van der Aalst. Analyzing resource behavior using process mining. In Stefanie Rinderle-Ma et al., editor, *BPM Workshops*, volume 43 of *LNBIP*, pages 69–80. Springer, 2009.
13. John R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
14. Anne Ratzer et al. CPN tools for editing, simulating, and analysing coloured petri nets. In *ICATPN 2003*, volume 2679 of *LNCS*, pages 450–462. Springer, 2003.
15. Anne Rozinat et al. Process mining applied to the test process of wafer scanners in ASML. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 39(4):474 –479, 2009.
16. Anne Rozinat and Wil M. P. van der Aalst. Decision mining in ProM. In *Business Process Management*, volume 4102 of *LNCS*, pages 420–425. Springer, 2006.
17. Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes.* Springer, 2011.
18. Wil M. P. van der Aalst et al. Process mining manifesto. In Florian Daniel et al., editor, *BPM Workshops (1)*, volume 99 of *LNBIP*, pages 169–194. Springer, 2011.
19. Wil M.P. van der Aalst and Schahram Dustdar. Process mining put into context. *IEEE Internet Computing*, 16:82–86, 2012.
20. Voluntary Inter-Industry Commerce Solutions Association. Voluntary Inter-Industry Commerce Standard (VICS). Last accessed: 22 May 2012.
21. Christopher D. Wickens and Justin G. Hollands. *Engineering Psychology and Human Performance.* Prentice-Hall Inc., New Jersey, 3rd edition, 2000.
22. Paul F. Wilson et al. *Root Cause Analysis: A Tool for Total Quality Management.* ASQ Quality Press, 1993.
23. Ian H. Witten et al. Weka: Practical machine learning tools and techniques with java implementations, 1999.

A Validation with Synthetic Log

We use the case study of a Purchase Order process based on the Voluntary Inter-industry Commerce Solutions (VICS) (Voluntary Inter-industry Commerce Solutions) industry reference model [20]. The risk incident whose root cause(s) are to be mined is the deadline expiry event (i.e. the *response* variable), using a derivable *predictor* variable, namely the resources' workload information detailed in Section 3.

A synthetic log was generated from the corresponding model of the Purchase Order process (modelled using the formal language of Coloured Petri Nets (CPN) [9] and aided by the CPN Tools [14]). The MXML-log-generator plug-in [11] was used to generate an MXML-formatted log file that we used to evaluate our approach to root cause mining. The details of the approach validation are provided in the remainder of this section.

A.1 Purchase Order Scenario - CPN Model and MXML Logs

Figure 5 shows the CPN model from which the synthetic log (used for the validation of our approach) was generated. This process consists of four manual activities: create purchase order (`CREATE PO`), approve purchase order (`APPROVE PO`), modify purchase order (`Modify PO`), and confirm purchase order (`CONFIRM PO`). This process also consists of two automated activities: order timeout (`ORDER TIMEOUT`) and process termination activity (`TERMINATE ORDER PROCESS` (see Figure 5). Since they are automated, the model assumes that they are completed as soon as they are assigned. When a transition representing a process activity (those transitions with thicker border in Figure 5) is fired, a corresponding code-region code is executed to add relevant event log entry to the log file (for example, see the *code region* for the transition `TERMINATE ORDER PROCESS` in bottom-right of Figure 5).

Figure 6 shows the details of the manual activity model (the same model is instantiated for all manual activities in the model). This model captures three activity lifecycle events: the *assignment/allocation* of an activity to a resource, the *start* of the activity execution by the assigned resource, and the *completion* of the activity by the resource. The model logs the occurrence of each of these events in the *code region* (not shown in Figure 6) of the corresponding activities (`‘ALLOCATE TASK’`, `‘START TASK’`, `‘COMPLETE TASK’`) .

A resource can be allocated with an activity at any time, regardless of whether the resource is ‘busy’ or ‘free’. As the resource starts to execute an activity, its status is set to ‘busy’ and it cannot execute other assigned activities. Upon the completion of the activity, the resource’s status is set to ‘free’ again. The duration of the execution of each manual activity is set to follow a normal distribution pattern. These rules are collectively expressed in the arc inscriptions and transition guards shown in Figure 6.

We configured our model to simulate the execution of 129 process instances. At the end of the simulation, 129 log files were generated (one for each process instance). We then used the ProM Import Framework tool [6] to merge those files into a single MXML file to be used for our subsequent analysis.

A.2 Enriching Event Log with Workload

We have implemented the log enrichment process using the synthetically-generated log. Firstly, we transformed the generated MXML file (from the CPN model) into a form that is suitable for relational algebra transformation, such as SQL queries. To this end, a raw event log table (called `eventlog` table) was created using the MySQL database system and the MXML log file was then imported into the table. A snippet of the MySQL table is shown in Table 4.

Once the log data was in a proper form, we applied the worklist derivation functions (defined in Section 3). The generated workload table is shown in Table 5.

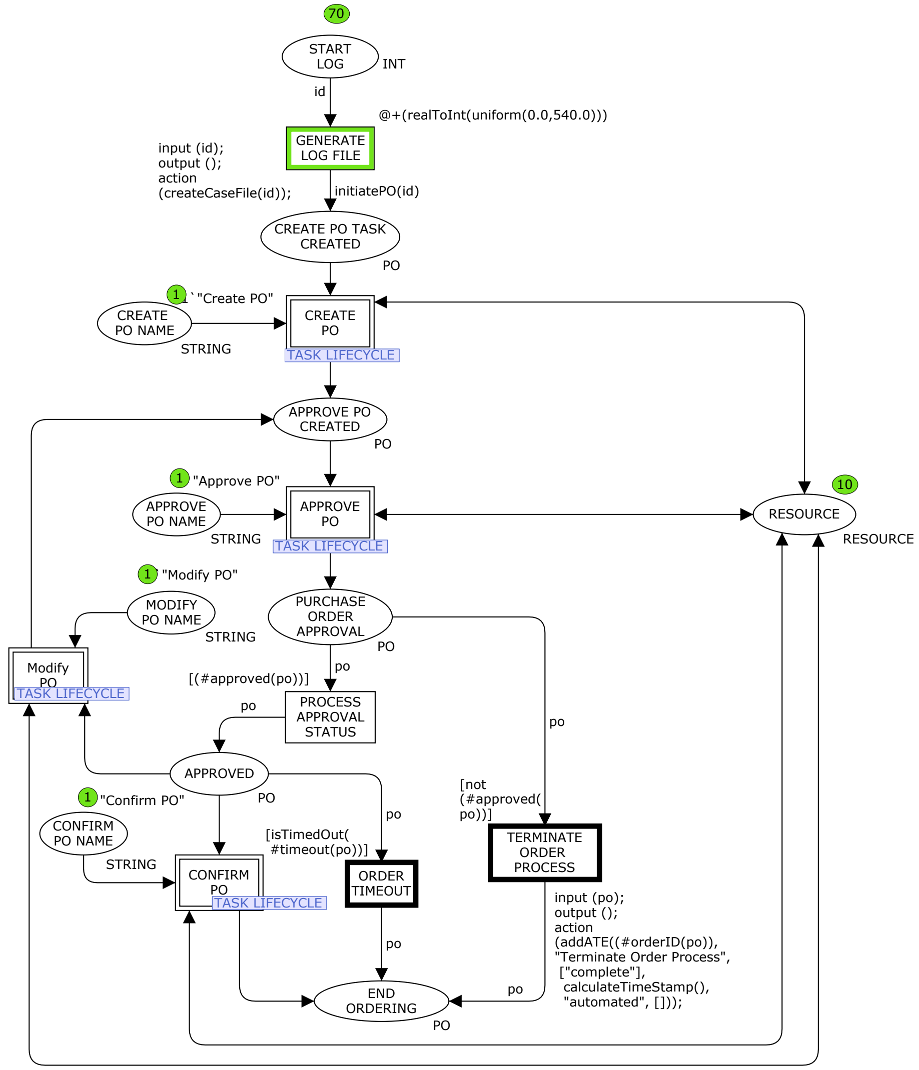

Fig. 5: CPN Model of a Purchase Order Process

A.3 Transforming Enriched Event Log to Classification-ready Case Log

The aggregation functions to derive the start time of a case, the end time of a case, the resources involved in a case, as well as the average workload of each resource as defined in Section 3 have also been implemented.

For a classification algorithm to work, the *response* variable must also be properly expressed in a categorical format. Therefore, we have also implemented the `getIsOverdue` function (defined in Section 3). The `threshold`

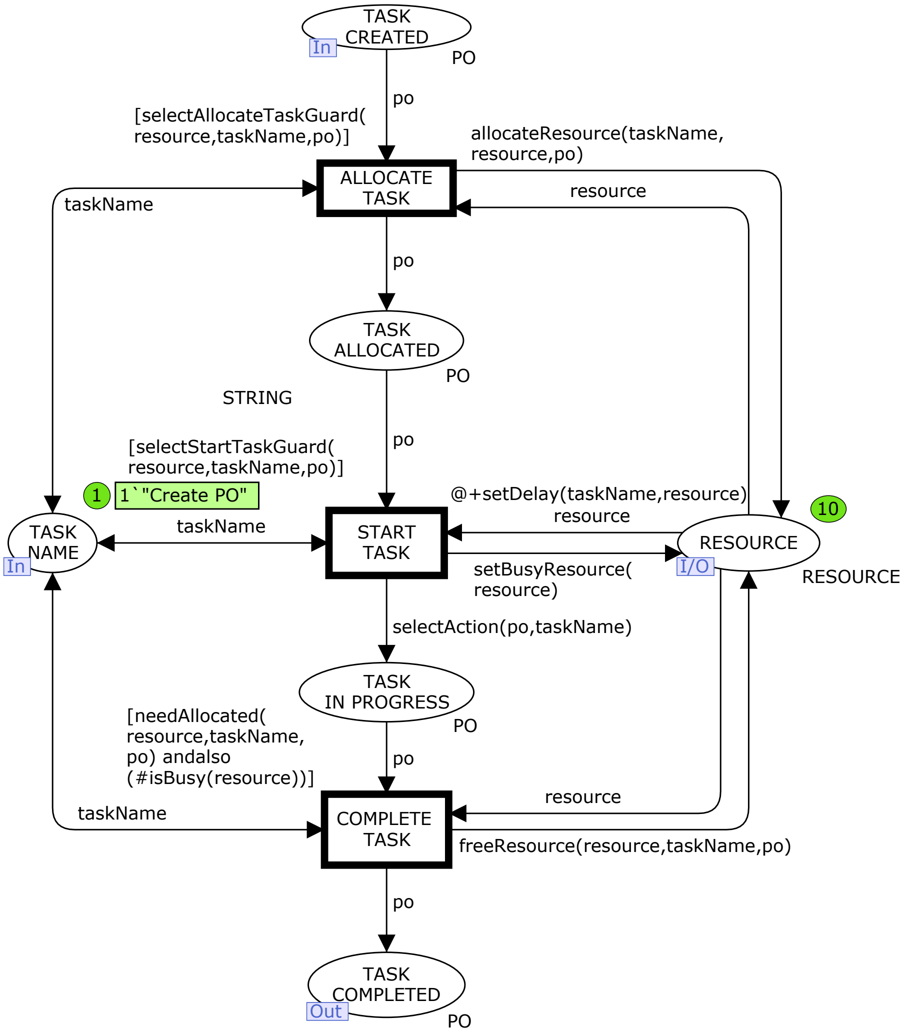
TASK CREATED
In
PO
po
[selectAllocateTaskGuard(resource,taskName,po)]
allocateResource(taskName, resource,po)
ALLOCATE TASK
taskName
resource
po
TASK ALLOCATED
PO
STRING
po
[selectStartTaskGuard(resource,taskName,po)]
1
1`"Create PO"
@+setDelay(taskName,resource)
resource
10
TASK NAME
In
taskName
START TASK
RESOURCE
I/O
setBusyResource(resource)
RESOURCE
selectAction(po,taskName)
TASK IN PROGRESS
PO
[needAllocated(resource,taskName, po) andalso (#isBusy(resource))]
po
taskName
resource
COMPLETE TASK
freeResource(resource,taskName,po)
po
TASK COMPLETED
Out
PO

Fig. 6: CPN Model of Purchase Order Manual activity Lifecycle

value is set to 30 hours in this case study. Similarly, the involvement or non-involvment of each resource should be presented as a distinct *predictor* variable in the classification-ready case log. Therefore, we have also implemented the `getIsResourceInvolved` function defined in Section 3. A snippet of the final classification-ready case log is provided in Table 6

A.4 RCA with Classification Techniques

Theoretically, by following the event log transformation process described so far, the data that reside in Table 6 should now be usable for various decision tree

caseID	**activity**	**eventType**	**timestamp**	**resource**
1	Create PO	assign	1970-01-01 18:14:00	8
1	Create PO	start	1970-01-02 22:31:00	8
1	Create PO	complete	1970-01-02 23:22:00	8
2	Approve PO	assign	1970-01-02 23:22:00	3
1	Approve PO	start	1970-01-02 23:22:00	3
4	Approve PO	complete	1970-01-02 23:30:00	3
8	Modify PO	assign	1970-01-02 23:40:00	10
...	...	...	...	...

Table 4: A snippet of imported MXML log into the `eventlog` MySQL database

`timestamp`	`r1`	...	`r2`	`r3`	...
1970-01-01 23:39:00	(103:Confirm PO),(3:Modify PO), (56:Confirm PO),(126:Create PO)	...	(54:Approve PO)	(104:Create PO)	... ...
...	...	...	...	...	...

Table 5: A snippet of the `workloadstarted` table

`caseID`	`start`	`end`	`isOverdue`	`isInvolved`$_{r1}$	...	`avgWL`$_{r1}$	`avgWL`$_{r2}$	...
95	1970-01-01 11:38:00	...	1	`false`	...	1.66427	0.988299	...
116	1970-01-01 12:38:00	...	1	`false`	...	1.61429	0.957895	...

Table 6: A snippet of a *classifier-ready* case log.

learning algorithms. As expected, the data imported data from the classification-ready `caselog` table into the data mining tool (called WEKA [23]) tool can be used to perform many *classification* analysis, including J.48, LADTree, Simple-Cart, and many others.

A representative example of the results of the decision tree learning analysis (using the J.48 decision tree learning algorithm [13] over 10-fold cross validation mode) is shown in Figure 7 (top part). In the beginning, we only used the average resource workload of each resource as the predictor variables. The accuracy of the result obtained is quite high: ≈ 86% true positive (TP) rate for 'on-time' cases (that is, the proportion of all 'on-time' cases that were correctly classified as 'on-time'), and ≈ 13% false positive rate for the 'on-time' class (that is, the proportion of all 'overtime' cases which were incorrectly classified as 'on-time'). Table 8 shows the full accuracy measures of the generated J.48 classification tree. This result may suggest that there were some significant correlation between average workload of resources and the occurence of 'overtime'/long-running cases.

Next, we included the information regarding the involvement (or non-involvement) of every resource in the analysis. Nevertheless, the result obtained (through the application of the same J.48 algorithm) is similar to the previous one (see Figure 7 - bottom part). Most importantly, the accuracy of the result is only slightly better.

Finally, we excluded the average workload information to see if the information regarding the involvement/non-involvement of resources is enough to explain long running cases. Using the same algorithm, the result obtained was

```
J.48 Classification Tree - CPN Log - only average workload - 10 fold cross validation
=====================================================================================
avgWL_r8 <= 2.78346
|   avgWL_r6 <= 0.989781
|   |   avgWL_r10 <= 0.978002: 1 (3.0)
|   |   avgWL_r10 > 0.978002
|   |   |   avgWL_r1 <= 1.44237: 0 (12.0)
|   |   |   avgWL_r1 > 1.44237
|   |   |   |   avgWL_r1 <= 1.49053: 1 (3.0)
|   |   |   |   avgWL_r1 > 1.49053
|   |   |   |   |   avgWL_r10 <= 0.997361: 1 (4.0/1.0)
|   |   |   |   |   avgWL_r10 > 0.997361: 0 (16.0/2.0)
|   avgWL_r6 > 0.989781: 1 (47.0/6.0)
avgWL_r8 > 2.78346: 0 (44.0)
```

```
J.48 Classification Tree - CPN Log - average workload + resource involvement
10-fold cross validation
============================================================================
avgWL_r8 <= 2.78346
|   avgWL_r6 <= 0.989781
|   |   resource7 = 0
|   |   |   avgWL_r8 <= 1.90742: 1 (3.0/1.0)
|   |   |   avgWL_r8 > 1.90742: 0 (17.0)
|   |   resource7 = 1
|   |   |   resource6 = 0
|   |   |   |   resource8 = 1: 1 (4.0)
|   |   |   |   resource8 = 0
|   |   |   |   |   resource2 = 0
|   |   |   |   |   |   avgWL_r1 <= 1.59603: 0 (6.0/1.0)
|   |   |   |   |   |   avgWL_r1 > 1.59603: 1 (2.0)
|   |   |   |   |   resource2 = 1: 1 (2.0)
|   |   |   resource6 = 1: 0 (4.0)
|   avgWL_r6 > 0.989781
|   |   avgWL_r3 <= 1.64218: 1 (45.0/4.0)
|   |   avgWL_r3 > 1.64218: 0 (2.0)
avgWL_r8 > 2.78346: 0 (44.0)
```

Fig. 7: J.48 Classification Tree - Analysis Results

much poorer: ≈ 68% TP rate (on average), with false positive rate going up to ≈ 37% (on average). Figure 9 shows the generated classification tree and the full accuracy metrics.

Based on these analysis results, we may say that workload is the main root cause for explaining long running cases.

By showing the details of our validation process, we have thus *demonstrated the* applicability *of our approach in transforming a process mining problem into a problem that can be solved using a relatively rich and matured classification techniques* from the field of data mining.

```
Accuracy Measures
=================
Correctly Classified Instances         112               86.8217 %
Incorrectly Classified Instances        17               13.1783 %
Kappa statistic                          0.7287
Mean absolute error                      0.1901
Root mean squared error                  0.3566
Relative absolute error                 39.4584 %
Root relative squared error             72.6615 %
Total Number of Instances              129

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.865     0.13       0.818     0.865     0.841      0.851    1
                 0.87      0.135      0.905     0.87      0.887      0.851    0
Weighted Avg.    0.868     0.133      0.87      0.868     0.869      0.851

=== Confusion Matrix ===

  a  b   <-- classified as
 45  7 |  a = 1
 10 67 |  b = 0
```

Fig. 8: Accuracy Measures for the J.48 Classification Tree (shown in Figure 7)

```
J48 pruned tree - Resource Involvement only
------------------------------------------

r7 = 0: 0 (62.0/17.0)
r7 = 1
|   r2 = 0
|   |   r10 = 1: 0 (17.0/4.0)
|   |   r10 = 0
|   |   |   r3 = 1
|   |   |   |   r4 = 1
|   |   |   |   |   r6 = 0: 1 (5.0/1.0)
|   |   |   |   |   r6 = 1: 0 (2.0)
|   |   |   |   r4 = 0: 0 (5.0)
|   |   |   r3 = 0: 1 (20.0/6.0)
|   r2 = 1: 1 (18.0/5.0)

Number of Leaves  :         7

Size of the tree :         13


Accuracy Metrics
================

Correctly Classified Instances          88               68.2171 %
Incorrectly Classified Instances        41               31.7829 %
Kappa statistic                          0.3204
Mean absolute error                      0.4045
Root mean squared error                  0.4821
Relative absolute error                 83.9634 %
Root relative squared error             98.2446 %
Total Number of Instances              129

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.519     0.208      0.628     0.519     0.568      0.625    1
                 0.792     0.481      0.709     0.792     0.748      0.625    0
Weighted Avg.    0.682     0.371      0.676     0.682     0.676      0.625

=== Confusion Matrix ===

  a  b   <-- classified as
 27 25 |  a = 1
 16 61 |  b = 0
```

Fig. 9: Generated J.48 Classification Tree and the Accuracy Metrics - Resources Involvement as Predictor Variable