UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Andrii Rozumnyi

A Dashboard-based Predictive Process Monitoring Engine

Master's Thesis (30 ECTS)

Supervisor: Ilya Verenich, MSc

Tartu 2017

A Dashboard-based Predictive Process Monitoring Engine

Abstract:
Process monitoring forms an integral part of business process management. It involves activities in which process execution data are collected and analyzed to measure the process performance with respect to the performance objectives. Traditionally, process monitoring has been performed at runtime, providing a real-time overview of the process performance and identifying performance issues as they arise. Recently, the rapid adoption of workflow management systems with logging capabilities has spawned the active development of data-driven, *predictive* process monitoring that exploits the historical process execution data to predict the future course of ongoing instances of a business process. Thus, potentially deviant process behavior can be anticipated and proactively addressed.

To this end, various approaches have been proposed to tackle typical predictive monitoring problems, such as whether an ongoing process instance will fulfill its performance objectives, or when will an instance be completed. However, so far these approaches have largely remained in the academic domain and have not been widely applied in industry settings, mostly due to the lack of software support. In this thesis, we have designed and implemented a prototype of a predictive process monitoring engine. The developed solution, named *Nirdizati*, is a configurable full-stack web framework that enables the prediction of several performance indicators and is easily extensible with new predictive models for other indicators. In addition, it allows handling event streams that originate from multiple business processes. The results of the predictions, as well as the real-time summary statistics about the process execution, are presented in a dashboard that offers multiple alternative visualization options. The dashboard updates periodically based on the arriving stream of events. The solution has been successfully validated with respect to the established functional and non-functional requirements using event streams corresponding to two real-life business processes.

Keywords:
Business process management, Process Mining, Predictive Monitoring, Machine Learning

CERCS: P170 Computer Science, Numerical Analysis, Systems, Control

Esipaneelil põhinev ennustav protsesside jälgimise süsteem

Lühikokkuvõte:
Protsesside jälgimine moodustab keskse osa äriprotsesside juhtimisest. See sisaldab tegevusi, milles kogutakse ja analüüsitakse protsessi täideviimise andmeid, et mõõta protsesside tulemuslikkust, võttes arvesse soorituse eesmärke. Tavaliselt on protsesside jälgimist sooritatud käitluse ajal, võimaldades reaalajalist ülevaadet protsessi sooritusest ja tuvastades protsessi vaidlusküsimused nende tekkmise hetkel. Viimasel ajal logimisvõimetega töövoo juhtimise süsteemide laialdane omaksvõtt on loonud aktiivse andmetest ajendatud ennustava protsesside jälgimise, mis kasutab varasemat protsesside jooksutamise andmestikku, et ennustada käimasolevate äriprotsesside tulevikusuunda. Seega potentsiaalselt hälbiva protsessi kulgu saab ette ennustada ja lahendada. Tüüpiliste protsesside jälgimise probleemidega tegelemiseks on välja pakutud erinevaid lähenemisi, nagu kas parasjagu käiva protsessi instants vastab selle soorituse eesmärkidele või millal instantsiga lõpule jõutakse. Need lähenemised on siiski seni jäänud akadeemilisse valdkonda ning neid pole rakendatud tööstuse sätetesse. Selles lõputöös me disainisime ja teostasime ennustava protsessi jälgimise mootori prototüübi. Arendatud lahendus on konfigureeritav täis-pinu veebiraamistik, mis võimaldab mitme soorituse indikaatori ennustamist ja mida saab kerge vaevaga laiendada teiste indikaatorite jaoks uute ennustavate mudelitega.

Lisaks võimaldab see mitmest äriprotsessist pärinevate sündmusvoogude käsitlemist. Nii ennustuste tulemused kui protsesside täitmise reaalaja statistika kokkuvõtted kuvatakse esipaneelil, mis võimaldab mitut erinevat alternatiivset visualiseerimise valikut. Lahendus on kahte tõsielu äriprotsessi kasutades edukalt valideeritud, arvestades defineeritud funktsionaalseid ja mittefunktsionaalseid nõudeid.

Võtmesõnad:
Äriprotsesside juhtimine, Protsessikaeve, Ennustav jälgimine, Masinõpe

CERCS: CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Acknowledgements

I must express my very profound gratitude to my thesis supervisor Ilya Verenich for providing me with unfailing support and continuous encouragement throughout the process of researching and writing this thesis.

I greatly appreciate the feedback and continuous guidance from Professor Marlon Dumas (University of Tartu) and Professor Marcello La Rosa (Queensland University of Technology).

The study of the author of this thesis is supported by the Estonian Foreign Ministry's Development Cooperation and Humanitarian Aid funds.

Last but not the least, I would like to thank my fiancé for supporting me spiritually throughout writing this thesis and my life in general.

Contents

1 Introduction

1.1 Context and Motivation

Business processes are generally supported by workflow management systems (WFMS) which record data about every individual execution of a process, commonly referred to as a process instance or a *case*. These data can be extracted as event logs in order to perform various forms of process analytics. The produced logs at least include the identifications value of each case, and an ordered sequence of tasks executed as a part of that case along with their start and end timestamps.

One area of business process management that has recently benefited from a widespread adoption of WFMS is process monitoring and controlling, which includes a range of activities to measure how well the process performs with respect to the organization's performance targets. Process monitoring can be carried out retrospectively, based on the event logs that relate to a particular period. Alternatively, it can be performed as a continuous online activity based on events of currently running instances, thus providing a real-time overview of the process performance. However, this approach is reactive in the sense that errors and deviations are identified once they have already occurred.

Predictive process monitoring takes a step ahead by enabling process participants and analysts to continuously receive guidance to achieve process objectives and higher performance. By knowing the process will violate some objectives, e.g. it will finish late, process participants can take proactive actions to prevent the violations. In this way, cases that are forecasted to violate the performance targets will receive a higher priority, so that the violation does not actually occur. For instance, in a freight transportation process, if a delay in shipping time is predicted, process owner could decide to reschedule transport routes or utilize additional delivery vehicles [17].

Another example of applying predictive monitoring is "anticipatory shipping" used by Amazon [25]. The idea is to predict what customers are going to buy before they actually do so. Accordingly, goods are shipped to the nearest warehouse to wait until the actual order is placed. These predictions are based on a history of orders, created wish lists, product searches, and even how long a user hovered a cursor over an item. Thus, Amazon potentially reduces delivery costs by utilizing its supply chain more efficiently, while customers enjoy shorter delivery times.

From the lean manufacturing perspective, predictive process monitoring can be beneficial in waste elimination. Specifically, at runtime, process participants can be notified about a potential accumulation of waste, such as unnecessary transportations, overprocessing, the waste of motion, waiting and so on.

1.2 Problem Statement

Predictive process monitoring takes a set of historical process execution cases (i.e. completed cases) and an ongoing process case (i.e. trace prefix) as input and predicts desired properties of that case. Several approaches have been proposed to solve standard predictive process monitoring tasks, e.g. whether a running process instance will meet its performance objectives, or when will a case be finally completed. However, the corresponding software, released as either stand-alone tools or plugins for some process mining frameworks, is designed for experimentation purposes in the sense that it builds a particular predictive model and assesses its goodness-of-fit on a batch of test samples at given stages of a case execution. Therefore, it is not applicable for real-time scenarios wherein process workers require a *continuous* predictive support. Furthermore, despite the wide range of enterprise software for an offline and online process monitoring and controlling, to the best of our knowledge, none of them includes capabilities for real-time predictive support. Therefore, so far the predictive monitoring approaches have largely remained in the academic domain and have not been widely applied in industry settings.

1.3 Objectives

In this setting, the aim of the project is to design and evaluate a prototype of a predictive process monitoring engine that would allow users to view information about the status of ongoing cases graphically. Such information can be divided into two parts:

- Descriptive that provides users with the analysis of the current state of all ongoing cases, e.g. load analysis and resource analysis
- Predictive (e.g. prediction of the remaining time, prediction of compliance violations) that allows users to respond, in real-time, to potential process performance issues before they arise

As an input, the engine would consume a stream of events coming from a WFMS. The results of the predictions, as well as the real-time summary statistics about the process execution, will be presented in a dashboard that should offer multiple alternative visualization options. The dashboard should updates periodically, as new events arrive.

The engine should be a web framework (in other words piece of software which simplifies development of web application) for Predictive Monitoring. At the same time, it should be scalable and flexible to meet the demand of industrial usage.

The developed system will support its users – process workers and operational managers – in making more informed, data-driven decisions to guide the process execution in the desired direction.

1.4 Structure of the Thesis

The rest of the thesis is organized as follows. In Chapter 2 we provide an overview of the main theoretical concepts and techniques used in this work. Chapter 3 describes the detail of the design, architecture and major components of the proposed solution. Furthermore, it clarifies the detail of the procedures and steps taken in developing our software. Besides that, it describes the various tools, frameworks, and technologies that we used in implementing and finalizing the approach. Chapter 4 presents an experimental evaluation of the designed predictive process monitoring engine for the scenario of a bank loan application process. This thesis is concluded in Chapter 5 together with the proposed future research directions.

2 Background and related work

2.1 Business Process Management

2.1.1 Overview

Business Process Management (BPM) is "a body of methods, techniques and tools to discover, analyze, redesign, execute and *monitor* business processes" [9]. In this context, a business process is viewed as a collection of inter-connected events, activities and decision points that involve a number of human actors, software systems and physical and immaterial objects, and that collectively lead to an outcome that adds value to the involved actors.

Traditional BPM practice has mostly relied on manual data gathering techniques for processes discovering, analyzing, and redesigning [35]. For example, traditional approaches to process monitoring rely on runtime observations of process work by business process analysts or "post-mortem" (offline) analysis of process execution. These techniques, while being able to provide estimations of the process performance, such as error rates and branching probabilities, often fail to provide a complete picture, including the numerous deviations and exceptions that usually characterize the process execution at a daily basis.

Evidence-based BPM aims to solve the limitations of traditional "empiric" BPM practices by systematically analyzing data produced during the process execution to monitor the performance of business processes [35]. In other words, process execution decisions are based on the experience obtained through past process executions. Evidence-based BPM has recently gained notable interest, due to the widespread adoption of workflow management systems that have capabilities to collect and store data about process executions, as well as due to advances in process mining techniques.

2.1.2 Process Mining

Process mining is an interdisciplinary area of knowledge that serves as a bridge between traditional model-based process analysis, such as process simulation, and statistics-based analysis, such as data mining.

Process mining deals with the analysis of event records (logs) produced during a business process execution. These event logs include at least the identification value of each performed case and an ordered sequence of events executed as a part of that case along with their completion timestamp. Optionally, a log may contain other *event attributes* such as the start timestamps, the ID of a person who performed the event, its associated cost, etc. Furthermore, *case attributes* may be present. These attributes are the same for each event of a given case. For instance, in a loan application process, the requested loan amount does not usually change for a given application. Table 1 contains an excerpt of a hypothetical log from such a process.

Table 1. Extract of an event log.

Case	Case attributes		Event attributes			
ID	Channel	Amount requested	Event	Timestamp	Resource	...
235	Email	$800	T02	2016-01-10 9:13	Mark	...
235	Email	$800	T06	2016-01-10 9:14	Ann	...
241	Phone	$1200	T02	2016-01-10 9:18	Lisa	...
235	Email	$800	T10	2016-01-10 9:45	John	...
241	Phone	$1200	T05	2016-01-10 9:17	Mark	...
287	Email	$3500	T02	2016-01-10 10:40	Mark	...

The goal of process mining techniques is to derive potentially useful, non-trivial information from the event logs that enables business analysts to understand the various aspects of process behavior as it is reflected in the event logs. Standard process mining tasks include process performance monitoring, automated process discovery, process model repair and enhancement, process variant identification and deviance mining.

2.1.3 Process Monitoring and Controlling

One of the most common tasks in process mining is process monitoring and controlling. Herein, data related to the process execution are collected and analyzed to assess the process performance with respect to its performance criteria. [32]

Traditional approaches to process monitoring and controlling based on "post-mortem" (offline) analysis of process execution. This range of techniques is usually referred to as *process analytics*, or process intelligence. It takes as input a database of completed process cases that relate to a specific period of time and outputs process performance findings, such as identified bottlenecks. An example of a process analytics tool is an open-source framework *ProM*[1] that employs a plugin architecture to extend its core functionality. Other popular tools for process analytics include Celonis[2] and Fluxicon Disco[3].

Another approach to process monitoring includes observations of process work by process analysts at runtime, i.e. online. This family of techniques known as *business activity monitoring* takes as input an event stream, i.e. prefixes of ongoing process cases and outputs a real-time overview of the process performance, such as current process load or problematic cases and current bottlenecks. Tools with business activity monitoring capabilities usually present output in the form of reports and dashboards. Figure 1 provides an example of a work-in-progress report as produced by the Bizagi Business

[1] `http://www.promtools.org/`
[2] `http://www.celonis.com/en/`
[3] `https://fluxicon.com/disco/`

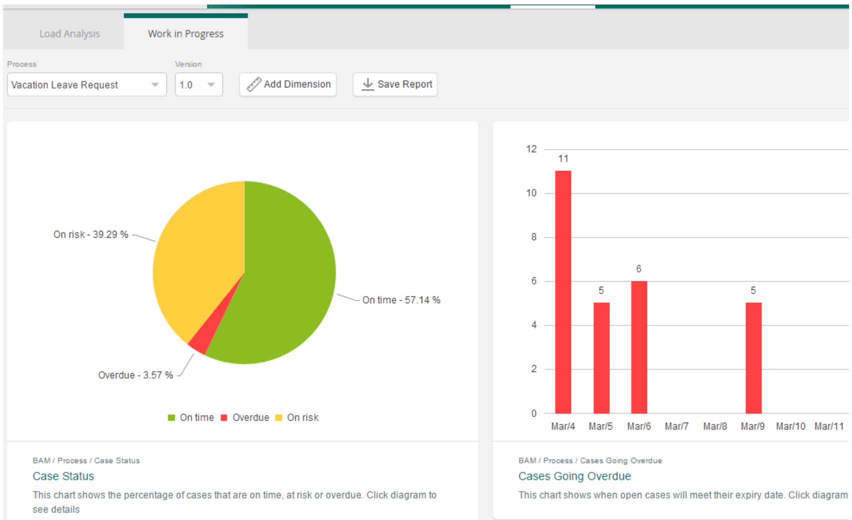


Figure 1. An example of a real-time work in progress report

Activity Monitoring tool [1]. The pie-chart on the left shows the percentage distribution of cases based on whether they are running on time, at risk, or overdue. The bar chart shows the target resolution date of ongoing cases.

The techniques in question are only able to detect process issues once those have arisen. The idea of *Predictive* process monitoring is to solve the issue of the limitations of traditional "empiric" monitoring practices through systematic analysis of data produced during the process execution [35]. In this way, predictive process performance can be viewed as an extension of the traditional business monitoring of activities. Such extension uses historical data so that those who participate in the process can control the execution of the process while taking precautionary measures to achieve in such a way the desired process objectives (Figure 2).

Every predictive process monitoring system has prediction models to rely on, which are built based on the event log of the already completed cases. In order to provide predictions of the future performance of the running cases, the earlier describes models are applied to prefixes. Participants of the process are notified by the issued alerts if the predicted result differs from what has been foreseen. In addition, process workers are assisted with recommendations regarding the mitigation of performance objectives violation. So, one can anticipate possible malfunctions and address them promptly.

2.1.4 Predictive Monitoring Approaches

In this subsection, classification of proposed techniques is done based on prediction goal they are solving, which includes time-related predictions, predictions of a case outcome

[1] `http://www.bizagi.com/`

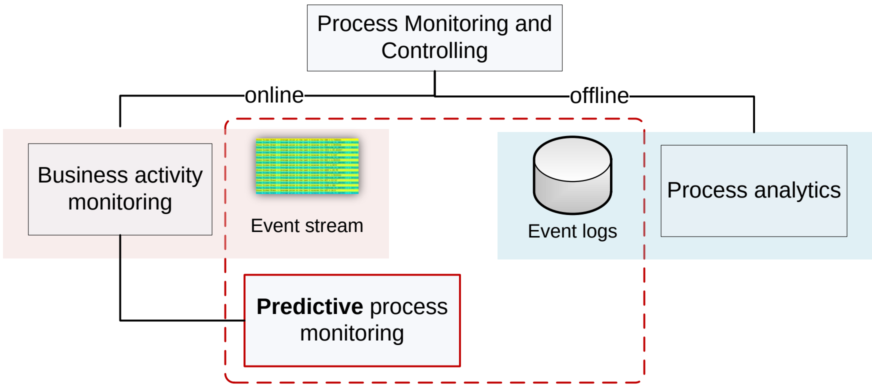


Figure 2. Process monitoring and controlling methods

and predictions of future events within a case.

Predictions of case completion time The case completion time is usually estimated via the prediction of *remaining* case processing time. Techniques in this space include van der Aalst et al. [28] who construct and apply an annotated transition systems using set, multiset and sequence abstractions of observed events. Rogge-Solti and Weske [22] model business processes as stochastic Petri nets and perform Monte Carlo simulation to predict the remaining time of a process instance. De Leoni et al. [7] propose a general framework to predict various characteristics of running instances, including the remaining time, based on correlations with other characteristics and using decision and regression trees. Verenich et al. [34] put forward a white-box approach for the remaining time prediction by estimating the processing time at the level of individual activities, and then aggregating these estimations using flow analysis techniques. The remaining time prediction problem has also been extensively studied in the context of software development processes. For example, Rees-Jones et al. [21] predict issue resolution time in Github projects using both project's intrinsic features as well as and cross-project contextual features.

Predictions of deadline violations Metzger et al. [16] present techniques for predicting "late show" events (i.e. delays between the predicted and the actual time of arrival) in a freight transportation process by finding correlations between "late show" events and external variables related to weather conditions or road traffic. Senderovich et al. [24] apply queue mining techniques to predict delays in case executions.

Predictions of process outcome Another category of techniques aims to predict the outcome of running cases. For example, Maggi et al. [14], propose a framework to predict the boolean outcome of a case (normal or deviant) based on the sequence of activities executed in a given case and the values of data attributes of the latest completed activity. Teinemaa et al. [27] construct one offline classifier for every possible prediction point (e.g. predicting the outcome after the first event, the second one and so on). Conforti et al. [6] apply a multi-classifier (decision trees) at each decision point of the process, to predict the likelihood of various types of risks, such as cost overruns and deadline violations.

Predictions of future events Lakshmanan et al. [12] use Markov chains to estimate the probability of future execution of a given task in a running case. Breuker et al. [5] use probabilistic finite automata to predict the next activity to be performed while Tax et al. [26] predict the entire continuation of a running case as well as timestamps of future events using long short-term memory (LSTM) neural networks. Van der Spoel et al. [29] predict the case continuation by traversing the shortest path in a process graph.

2.2 Data Mining and Machine Learning

Predictive process monitoring is a multi-disciplinary field whose quick development is mostly due to the advances in the areas of data mining and machine learning. This section gives an informal introduction to machine learning and process mining and summarizes state of the art of predictive process monitoring.

2.2.1 Overview

Machine learning is a research area of computer science, concerned with the discovery of models, patterns, and other regularities in data [18]. Closely related to machine learning is data mining. Data mining is the "core stage of the *knowledge discovery process* that is aimed at the extraction of interesting – non-trivial, implicit, previously unknown and potentially useful – information from data in large databases" [10]. Data mining techniques focus more on exploratory data analysis, i.e. discovering unknown properties in the data, and are often used as a preprocessing step in machine learning to improve model accuracy.

A machine learning system is characterized by a learning algorithm and training data. The algorithm defines a process of learning from information extracted, usually as *features vectors*, from the training data. In our project we will deal with *supervised* learning, meaning training data is labeled, i.e. represented in the following form:

$$D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) : m, n \in \mathbb{N}\}, \tag{1}$$

where $\mathbf{x}_i \in \mathcal{X} = \mathbb{R}^m$ are the feature vectors and $y_i \in \mathcal{Y} = \mathbb{R}^n$ are the corresponding class labels.

A framework uses the feature vectors extracted from the labeled training data to build a predictive model that would assign labels on new data given labeled training data while minimizing error and model complexity. In other words, a model generalizes the pattern, providing a mapping $\mathcal{X} \to \mathcal{Y}$. The class labels can be either continuous, e.g. cycle time of activity, or discrete, e.g. loan grade. In the former case, the model is referred to as regression; while in the latter case we are talking about classification model.

To compare models, there should be chosen one of the available metrics. In a case of a regression problem Mean Absolute Error (MAE) is frequently used:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{2}$$

where $y_i \in \mathcal{Y} = \mathbb{R}^n$ is an actual value of a function in a given point and $\hat{y}_i \in \mathcal{Y} = \mathbb{R}^n$ is a predicted value.

In a case of classification problem, the common metric is an area under the curve (AUC). In such a case a curve is a receiver operating characteristic (ROC) which shows the trade-off between true positive and false positive rates for a binary classifier. The area under a ROC curve represents the probability that an arbitrarily picked example of a positive class will be ranked higher than an arbitrarily picked negative one. An area of 0.5 is equivalent to random guessing, and an area of 1 is the perfect classifier.

From a probabilistic perspective, the machine learning objective is to infer a conditional distribution $P(\mathcal{Y}|\mathcal{X})$. A standard approach to tackling this problem is to represent the conditional distribution with a parametric model, and then to obtain the parameters using a training set containing $\{\mathbf{x}_n, y_n\}$ pairs of input feature vectors with corresponding target output vectors. The resulting conditional distribution can be used to make predictions of y for new values of $\mathbf{x}$.

2.2.2 Ensemble Methods

In this work, we apply ensemble methods as they are simple to understand and implement yet powerful enough, and have been used for many predictive monitoring problems [13, 27, 33].

The objective of ensemble methods is to aggregate the results from several less accurate estimators to improve generalizability and obtain a model with higher accuracy.

There are two primary groups of ensemble methods:

- The main principle of **averaging methods** is to construct several estimators separately and to calculate their average predictions. Due to a reduced variance, the aggregated estimator proves to be better than any of the single base estimators. Among examples, one can list bagging methods and forests of randomized trees.

- On the contrary, in **boosting methods**, base estimators are successively constructed, and one attempts to mitigate the bias of the combined estimator. So, there is a motivation to combine weaker models in order to generate a powerful ensemble. One can name AdaBoost and gradient tree boosting as examples.

Random forests method implies that each decision tree in the ensemble is constructed from a bootstrap sample from the training set, i.e. random subset of data with a possible repetition of data points. At the same time, the tree is built on the top of random features subset. Such randomness causes a slight increase of the forest bias; however, as a result of averaging there is a decrease in its variance which is usually enough for getting a better model.

In gradient boosted trees (GBM) a generalization of boosting is performed with arbitrary differentiable loss function. Given a high accuracy of GBM for both regression and classification problems, it is widely used as the prediction model. However, there is a downside to be taken into account. Due to the successive nature of boosting, GBM it is not applicable for concurrent execution.

In this work we use random forest and gradient boosting, as implemented in the `scikit-learn` library [20] for `Python`. To choose the values for the main training parameters, such as the number of trees or learning rate, we apply hyperparameter optimization and cross-validation.

2.2.3 Feature Encoding

Naturally, business processes are represented as ordered sequences of events each having associated event attributes, i.e. so-called complex symbolic sequences [33]. Each case can then be abstracted as a sequence trace. However, as machine learning require an input in the form of feature vectors, the process sequences first need to be transformed, or *encoded* as feature vectors.

Leontjeva et al. [13] studied many different encoding methods. Some of them contain only partial information from the sequence, for example, *frequency-based* encoding only includes the number of occurrences of each event type in the sequence, but not the order in which they have happened. In comparison, *index-based* encoding preserves all the information contained in the original sequence so that it is possible to restore the sequence from a feature vector.

2.2.4 Stream Mining

In many real-life processes, data continuously accumulate over time at a rapid rate, specifically in areas such as telecommunications, financial markets, web applications, security systems, etc. These dynamic datasets composed of transient chunks of data are commonly known to as *data streams* [11].

Such data streams pose a number of unique challenges that make working with them different from working with static datasets of a finite size [2]:

- As the streams are potentially unlimited in size, it is supposed that the data are not archived for the future processing and can be processed *only once*. Thus, random access to data assumed by many machine learning and data mining algorithms may not be possible.
- The underlying data patterns are usually stationary and evolve over time. This aspect of data streams is referred to as *concept drift*.
- The stream is usually generated by an external process. Thus, a user might not control the arrival rate of the events. When the arrival rate varies with time, it may be difficult to perform data processing during peak periods.

To address these challenges, many stream data processing methods have been proposed. These methods include but are not limited to [2]:

- Reservoir sampling that aims is to *continuously* maintain a *dynamically* updated sample of k points from a data stream instead of trying to store the whole stream.
- Sliding windows that capture recent events in a timeframe of a fixed or variable size w and use it make decisions upon.
- Synopsis construction where the goal is to construct a higher-level abstraction of the part of a data stream to use it for mining.
- Some application-specific algorithms have been designed to cluster data streams, detect outliers in streams and perform streaming classification [2].

3 Approach and Implementation

In this section, we analyze requirements for the software solution to be developed in the first place. Then, the proposed system design is described in detail. As the next step, all technologies which are needed for implementation of the proposed architecture are listed and explained one at a time. Finally, special features of implemented software have been mentioned.

3.1 Requirements Analysis

Requirements for the predictive process monitoring engine originate from the primary users of such solution – process workers and operational managers. These requirements can be grouped into functional and non-functional. Let us enumerate *functional* requirements describing core functionality of the future solution:

FR1: Future case behavior The system must provide forecasts regarding the future course of ongoing process cases; for example, expected case completion time, probability of an error occurrence, most likely continuation (suffix) of a case, etc.

FR2: Current case behavior The system must provide at least the most basic information about the cases, such as case identification value, whether the case has completed or not, which events have occurred so far in a given case, when the case has been started, and the arrival time of the most recent event.

FR3: Visualization options As the amount of data in a real system is often quite big, it is important to have a possibility to control which data is present and how it is visualized. Thus, in a case of a table, it should be possible to control which columns are visible. In addition, pagination mechanism should be available. Coloring table rows and separate cells based on condition may serve as an additional informative dimension for data representation. Thus, rows can be colored differently based on whether a case is finished or not. The important feature is to draw the user's attention to values which have undesired outcomes, slow duration, violation of logical rule, etc. This can be done by color-coding.

FR4: Data sorting It should be possible to sort cases by specific attribute or feature.

FR5: Data query The system must have a basic search functionality as a number of cases in a real-life scenario might grow very fast. It stimulates the necessity to quickly filter out a particular group of cases and later make analysis over that subset of instances.

FR6: Data export The export functionality must be present in the developed software to give users a possibility to perform tailor-made data analysis and visualization.

FR7: Aggregated process state The system must provide a high-level overview of the current process load and performance. This may include the number of running and so far completed cases, the number of occurred events, average number of events per case and average case duration.

FR8: Data representation The system must be able to represent the process execution data using multiple alternative views. Apart from the default table view, the system should show the distribution of cases with respect to a specific performance objective, the distribution of cases by the case duration, by the remaining processing time, as well as by the number of events per case (case length). For running cases, such information must be predicted, while for the already completed cases the actual data should be used.

FR9: Multiple users Multiple users must be able to interact with the system simultaneously, without interfering with each other.

Next, we elaborate on the *non-functional* requirements detailing the system performance and quality:

NFR1: Platform support To easier accommodate for a growing range of operating systems and hardware varieties, a system should preferably be implemented as a *web* application according to a responsive design concept. Then it will be able to support a wide variety of devices as long as they have networking capabilities.

NFR2: Stream processing The system should be able to work with real-time event streams coming from, for instance, a workflow management system. This imposes severe restrictions on the runtime performance of the system. In general, the system should process new events faster than the mean interarrival time of events.

NFR3: Multiple streams The system should be able to handle event streams coming from multiple sources. Moreover, it should be able to handle different streams both from the same and from different business processes. Such system design allows developing custom solutions with the same server setup. Simultaneous handling of different business processes makes it a crucial necessity for configuration possibility. Each business process might have its predictive models; at the same time, a number and type of models may vary.

NFR4: Scalability The system design architecture should be horizontally and vertically scalable, i.e. there should be a clear strategy on how to handle the bigger traffic load.

NFR5: System configuration As the first iteration, all configurations can be specified manually in a structured file in a JSON or YAML format. Later, it should be

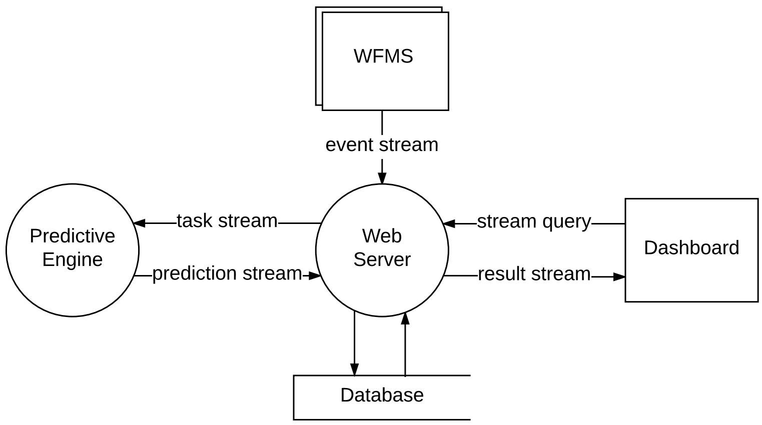


Figure 3. High-level data flow diagram in a Yourdon-Coad notation

replaced by the possibility for the end user to manage settings from UI side which will be stored in a database.

NFR6: Model extensibility The system should be easily extensible with new predictive models. Moreover, such scripts may be written in any programming language. They should take a partial trace as an input and return the predicted values.

3.2 System Design

As a preliminary step to conceive a required system architecture, let us illustrate the flow of data through the proposed system. As noted earlier, a workflow management system (WFMS) continuously registers tasks performed in the organization. This serves as an input to our system, in the form of a stream of events. Next, the produced stream is consumed by a web server which is responsible for storing data and further information processing. Web server, in turn, communicates with a predictive engine which applies pre-trained models for a business process and returns all the results of calculations. After web server receives the outputs from each predictive model, it writes them to a database and updates corresponding clients. From the user's side, the results are visualized via a dashboard-based interface which is capable of sending queries requesting various visualization options.

Based on this data flow diagram, we devise the main parts and high-level components of the system (Figure 4):

- streaming platform
- web server with load balancer and internal components
- predictive engine

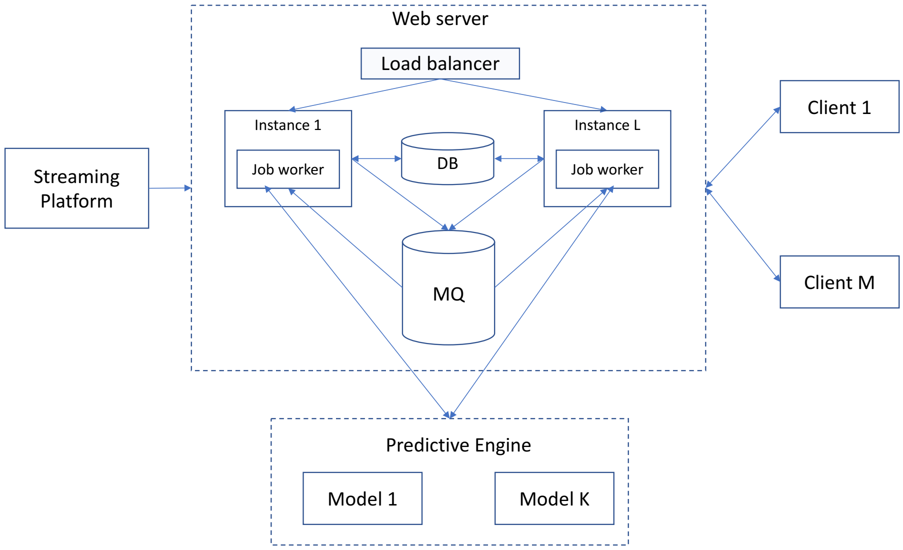


Figure 4. Proposed system design architecture

- client-side user interfaces

Let us look at the architecture in more detail. The streaming platform is responsible for transferring events from different sources to the web server, which, in turn, processes incoming events and updates corresponding clients. The right choice of a streaming platform allows handling high-load traffic, any number of sources, and business processes. Possibility to process big amounts of messages requires ability for easy scaling.

Proposed design for streaming platform includes producers of real-time events, cluster for temporary messages storage, and consumers. The streaming platform should support grouping of messages. Such groups are called topics. One topic can correspond to one business process. Consumer subscribes from one to many topics. Depending on the requirements and resources availability, it can be more than one consumer. After consumer retrieves the message from the cluster, it is transferred further to a web server. Thus, in such a way streaming platform can handle messages not only from several sources of the same process but from different processes. The same can be achieved by adding information about business process into a message payload.

When a message is received by a web server, load balancer should define which internal server instance will be responsible for handling that message. In such system, it makes sense to use balancer based on resource availability of instances. Such usage of the load balancer would be possible if server instances have stateless architecture design, which means that each message is processed completely independently and does not depend on previous. Thus, with such web server setup, the system will use all available resources and equally divide the load between internal server instances.

When the message is reached internal server instance, the system state is updated based on received information by writing and modifying data in the database. At the same time, all clients observing a particular business process, get notified about the new event, and UI is updated accordingly. After that, a job is ready to be published into a message queue. For this, we need to retrieve partial trace of a case which contains received event.

Every server instance contains a job worker. It subscribes to new jobs in the message queue. Having received the message, worker sends it to the predictive engine. The primary role of the engine is to apply all predictive models which are trained for a business process of the received event. Calculations can be done on the same hosted server, on another server, or in a cloud computational service. The models present an outcome of various metrics, such as the remaining time of an instance, next activity to be performed, etc. With all the models being applied, results should be sent back to the job worker which, in turn, can trigger system update by writing them to the database. Eventually, all relevant clients are notified with new information.

The client-side architecture is built as a Single Page Application (SPA). It means that user interface consists of one web page; however, some parts of it might be hidden while the others are active. SPA mimics interaction with the system the same way as with desktop application. The client receives a single web page with all needed for correct work components with the first request to a server. Also, it is set up a bidirectional channel for communication between client and server. Eventually, the user can navigate to different parts of the webpage without any additional requests as all components have already been received and the communication channel is set. Such approach decreases response time and improves overall user experience.

3.3 Technology Stack

For a full-stack web application development, there has been used a wide range of modern technologies. They can be divided into several categories. Let us shortly describe them here.

3.3.1 Streaming Platform

The streaming platform is part and parcel of a reliable application which handles streams. Apache Kafka[1] (from now on referred to as Kafka) is one of the most popular enterprise solutions for that. It enables publishing and subscribing to records streams

[1] `https://kafka.apache.org/`

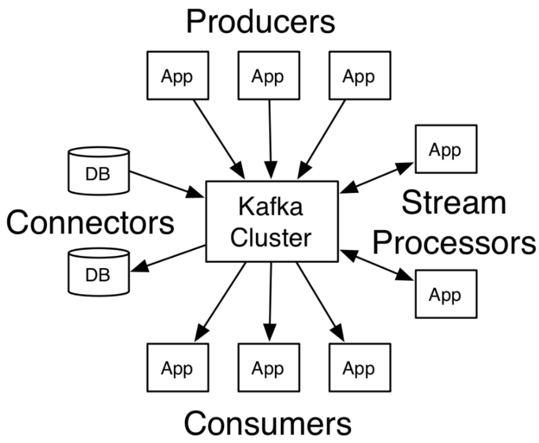


Figure 5. Apache Kafka distributed streaming platform architecture taken from [3]

with different topic support. Internally, each record consists of a key, value, and timestamp.

Kafka is highly scalable, efficient, reliable, and durable publish-subscribe messaging system. It is running on a cluster which can be spread to several servers since it is a distributed system. One of the strongest advantages of Kafka over other similar systems is high throughput for both publishing and subscribing. That is why it proves useful for applications with a high-loaded stream of messages. Kafka is reliable due to data replications. Also, it is able to persist messages on a disk storage.

Figure 5 describes general architecture for Kafka.

The documentation states that Kafka has four key components:

- The *Producer* enables message publishing to one or more topics.
- The *Connector* enables connecting Kafka to existing data systems such as database and interacting with them. For instance, it can track changes in a database and, be relying on that, trigger defined events.
- The *Streams* enables converting input streams of records to output.
- The *Consumer* enables subscribing to one or more topics and after that consumes published records by the publisher.

3.3.2 Web Server and Internal Components

As a web server, Nginx[1] has been chosen which is the fastest available nowadays. That is the reason why it is frequently used by high-traffic websites. Web server is used to handle HTTP requests. HTTP stands for Hypertext Transfer Protocol, and it is a basic

[1] `https://www.nginx.com/resources/wiki/`

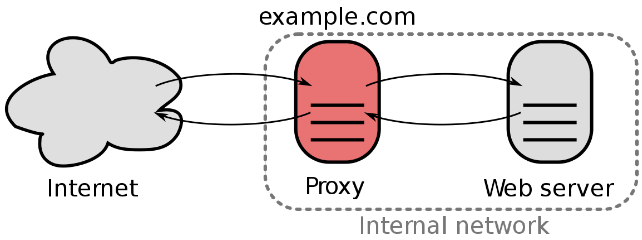


Figure 6. Reverse proxy web server architecture taken from [36]

protocol for communication in World Wide Web. Such requests can be grouped based on intended action[1]. The most common requests are:

- GET to retrieve a resource
- POST to create a resource
- PUT to update a resource
- DELETE to delete a resource

Other notable usages of Nginx are reverse proxying, load balancing, and caching.

The reverse proxy is the type of a web server which hides internal architecture. It transfers requests further to one or more servers on behalf of a client. After that, the reverse proxy server sends a response back as any other web server does (figure 6).

Nginx, working as the reverse proxy server, make use of a load balancing capability, i.e. it delegates requests to internal, namely, upstream web servers based on predefined rules. The balancing can be done based on the following mechanisms:

- in a round-robin fashion
- least number of active connections
- based on IP address

Besides, it is possible to specify weights for each internal server. Thus, the distribution of requests will be rebalanced taking that into an account.

The caching capabilities are needed to reduce the load on upstream servers. When a client requests a particular resource for the first time, Nginx may be configured in such a way that it will remember that resource. When the next client requests the same resource, then Nginx sends it straight away without asking from internal servers. It saves time by reducing the load of upstream servers.

Node.js[2] has been chosen as the primary programming language for development. It allows building highly scalable networked applications with a significant amount of

[1]https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
[2]https://nodejs.org/en/

requests and high traffic. The main reason which lies behind is that it has an event-driven architecture enabling writing of asynchronous programs.

The asynchronous approach prevents wasting time during execution of time-consuming input/output operations. Let us suppose that there is such an operation. Thus, to avoid wasting time till it is finished, the code needs to be segmented into two parts. The first one is dependent on the result, while the second one is independent. The second part can continue execution straight away, but the first part has to be wrapped into a function (callback function) which takes the result of the operations as a parameter. Later Node.js will trigger callback function using event after the operation is finished. To sum it up, a callback function is assigned to a particular operation, and after it is completed, the event will be triggered which causes the execution of callback function. [23]

Node.js is a server-side version of JavaScript language which is built on top of the highly efficient V8[1] engine which has been developed by Google and open sourced in 2008. JavaScript is an interpreted language which means that it is executed line by line by an interpreter. Thus, it is not possible to make low-level optimization for increasing the running speed of the program. Consequently, interpreted languages are slower than compiled ones. However, V8 engine improves efficiency by compiling JavaScript code into low-level machine code and execution is done in a similar way as with compiled languages. [23]

Even though V8 engine speeds up scripts running, Node.js should be used carefully in case of CPU intensive tasks as it uses single execution thread. It means that either programmer should adopt the task for asynchronous execution, or use separate back-end service for such tasks. Sometimes, the problem can be solved by increasing the number of server instances. Thus, the load can be spread across several servers.

On top of Node.js Express web framework[2] has been used for server-side development. Express is one of the most famous web application frameworks because it is minimal but at the same time flexible enough to build single as well as multi-page web applications by a small amount of code [23].

MongoDB[3] has been chosen as a data storage. It is NoSQL database which operates with data in JSON (JavaScript Object Notation) format which, in turn, is stored internally in BSON (Binary Structured Object Notation) format for efficiency increasing. The fact that MongoDB uses JavaScript interface simplifies the process of development, as the same language is used for client, server, and database layer [15]. Another benefit of using MongoDB is that it has no schema, which means that you do not need to specify the exact structure of the stored documents before they are stored. MongoDB does not require having a rigid schema structure for storing data. However, it can be done by

[1] `https://developers.google.com/v8/`
[2] `https://expressjs.com/`
[3] `https://www.mongodb.com/`

a developer at the application level. [23]

MongoDB supports index usage. It is a data structure which speeds up information search in a database at the expense of additional data storage. Using indexes, there is no need to examine every document in a database as they allow skipping most of them.

MongoDB is scalable and at the same time efficient which makes it applicable for heavy-loaded applications. MongoDB supports horizontal partition of data through sharding concept. In other words, it can scale not only due to more powerful hardware regarding RAM, CPU, and storage resources but by just adding more machines. Usually, the second option is much cheaper and has higher limits for data storing.

To simplify interaction with MongoDB, Mongoose[1] module has been used. Mongoose is the library which allows specifying schema in application layer which can be changed at any moment of time. The advantage of using it is built-in features such as validation of data type, the uniqueness of the value, or whether a field is required for storing the whole document or not. It adds the possibility to use virtual properties, i.e. fields which can be calculated on the fly based on data of the stored document. This feature helps to maintain data consistency. In addition to the reasons above, it allows using documents from the database as JavaScipt objects with methods for saving, finding, creating, and removing documents. [23]

As Mongoose is pretty flexible, it allows storing all information of the document even if some of the fields have not been specified in the schema. This enables saving additional fields to a database while taking an advantage from built-in validation only over specified fields.

Another component which helps to make an application scalable is message queues. As it can be inferred from the name, it is used for communication between different pieces of software. At the same time, it is served as a temporary storage for messages if messages cannot be handled by receiving component. The simplest architecture setup would consist of one or several sources of those messages, which are called producers. Producers not only generate messages but also put them into a queue. From another side, there are one or more receiving components – consumers. They are connected to the queue and process the message when received. A message itself contains the required information for execution of a particular task.

Message queues allow decoupling different components of architecture. Thus, each part can be developed independently or even replaced at some point by another component or service. For instance, in the developed software all calculations are done by hosted server of the application, however, in the future, those calculations can be shifted to cloud computational services or another dedicated server. If consumers cannot handle an amount of messages, in most cases this problem can be solved by increasing the number of instances. At the same time, as the queue can receive messages from different producers, the application can process messages from any number of sources.

[1] `http://mongoosejs.com/`

Implementation built on top of NoSQL database Redis[1] has been chosen for message queue functionality. According to the documentation, Redis Simple Message Queue[2] is a Node.js module for a lightweight message queue which does not require any dedicated queue server. That module allows to produce and consume more than 1000 messages per second on an average machine which is often enough even for a real enterprise system. Moreover, there is a guarantee that message will be delivered to exactly one recipient if message visibility timeout has not expired before.

socket.io[3] is another important technology which has been used in the developed software. It is JavaScript library which provides a real-time bi-directional event-based communication implementing WebSocket[4] protocol. Before channel of communication (socket) is established, a client sends a connection request to the server. Then, a channel is open till one of the sides closes it. At the same time, it is ready to transfer a message either from client to server or vice versa. It is worth mentioning that socket.io supports different rooms for communication, i.e. not related channels. Thus, it makes possible for a server to send messages to a particular set of clients which are grouped based on predefined rules.

For development and deployment simplification docker[5] containers have been used. This is a tool which is needed to encapsulate all dependencies of a piece of software inside a container. Such container stores everything that is essential for correct work of that encapsulated software. Moreover, Docker is independent of whether it runs on Windows, Linux or Mac OS. It is an abstraction which is built on top of OS. Docker containers ensure that everything works properly on any system setup. [19]

3.3.3 Predictive Engine

Figure 7 outlines the structure of a predictive engine to be used. Firstly, we select a historical set of completed process cases. Then we encode them as feature vectors. These feature vectors are then fed to machine learning classifiers to train a model that is saved on a disk for later use. At runtime, given an ongoing process case, we encode it with the same encoding method as for the training, and apply the previously saved model. The engine returns a prediction result, e.g. the remaining execution time or a probability of future deviance.

Python programming language has been chosen to implement the predictive engine functionality. It is one of the most common languages for data analysis in production. `Scikit-learn` [20] is the fast and efficient Python library that contains an implementation of popular machine learning techniques, including such ensemble methods as

[1] `https://redis.io/`
[2] `https://www.npmjs.com/package/rsmq`
[3] `https://socket.io/`
[4] `https://www.websocket.org/`
[5] `https://www.docker.com/`

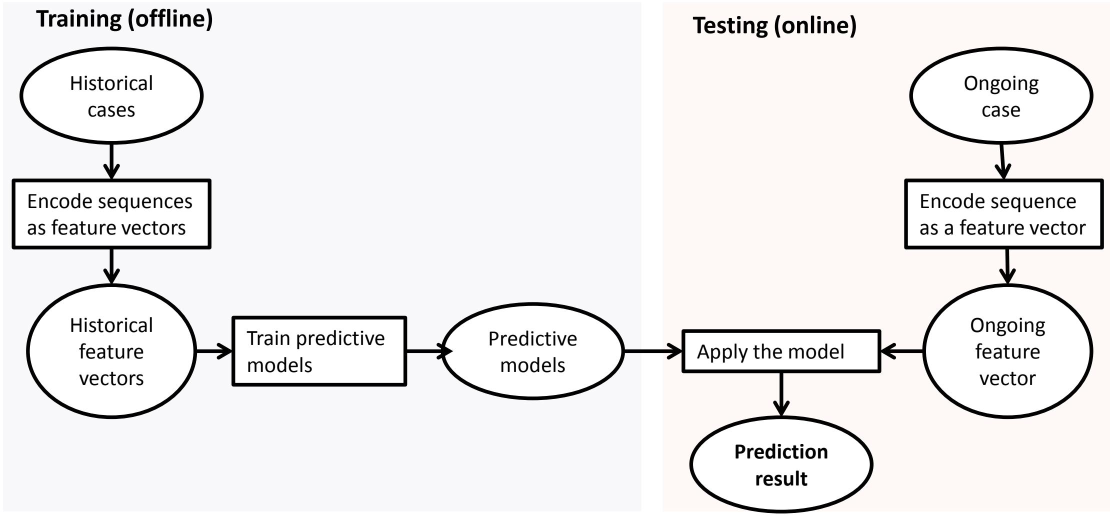


Figure 7. Structural overview of the predictive engine

random forest and gradient boosting. The produced models can be saved using `joblib` or `pickle` tools.

3.3.4 Client-side User Interface

With a big range and diversity of devices which can browse the Internet, the layout of web applications becomes a big problem. Web pages which are nicely rendered on big screens are not convenient to use on mobile devices and vice versa. The concept of responsive design is supposed to solve such problem. The idea that lies behind is that all elements of the web page are resizeable. Also, several layouts are used, each for different screen sizes. For instance, it can be done in three different designs: for mobile devices, tablets, and laptops. Therefore, you do not need to develop separate applications for each type of device. That is why, as a primary web development framework for client-side implementation, Bootstrap[1] has been used. As a framework, it contains reusable components of JavaScript and Cascading Style Sheets (CSS) files. JavaScript is used as a programming language for a client-side application and CSS is a stylesheet language which is responsible for visual representation of HTML document. The central idea behind Bootstrap is to enhance responsive design development. It can apply different layouts adjustable to screen size; at the same time, all broad range of reusable components are resizeable. When working with Bootstrap and CSS itself, it is recommended to use one of the preprocessors such as Sass[2] or Less[3]. They extend CSS by

[1] `http://getbootstrap.com/`
[2] `http://sass-lang.com/`
[3] `http://lesscss.org/`

adding new features which can be converted later to plain CSS. Such preprocessors are mainly used for writing cleaner CSS code.

Client-side development often includes small, but at the same time repetitive jobs such as minifying files, concatenation of them, CSS preprocessing, copying files to a specific folder, testing, deploying, etc. That is why it is reasonable to use task runners which automate repetitive tasks. The most popular JavaScript task runners are Gulp.js[1] and Grunt[2]. They are both used in production; so, it is just a matter of preference which one to choose. In current solution Gulp.js has been used.

3.4 System Implementation and Deployment

In this subsection, the details of an implementation of the developed software are outlined. Figure 8 describes architecture of implemented solution. The general idea of implementation is based on previously described system design with a few simplifications.

Due to the absence of real-time event stream, there was developed an execution engine or, in other words, replayer. The replayer streams an event log in the order of timestamps (i.e. events are streamed in the order of their arrival, with the interarrival time being equal or proportional to that of the log). Besides, the execution engine can send messages with constant interarrival time which simplifies software performance testing under different loads. The replayer mimics the whole streaming platform and sends HTTP POST requests to Nginx server. Thus, the web server does not detect any difference about whether messages have been sent using execution engine or a real-time event stream. One more important reason for the replayer development is the ability to test and debug the system. A developer should have a possibility to work on software in a local development environment. Another simplification for architecture is that a predictive engine is located on the same hosted machine. The rest of the workflow is the same as described in the subsection about system design architecture.

Initially, clients make an HTTP GET request to Nginx server which returns an HTML page back to them. After that, socket communication channel is set between the client and the web server. It worth mentioning that each client is assigned to a specific socket room. Each room is responsible for communication regarding the particular business process. The channel is defined either based on client's authentication, or on information that is received from the client initially.

As it was mentioned earlier, in technology stack overview, MongoDB has been chosen as a database storage. Database schema consists of two types of documents: Event and Case. Event document has the same structure as a message that came from the source, i.e. it has the same fields as the received message. The current implementation

[1] `http://gulpjs.com/`

[2] `https://gruntjs.com/`

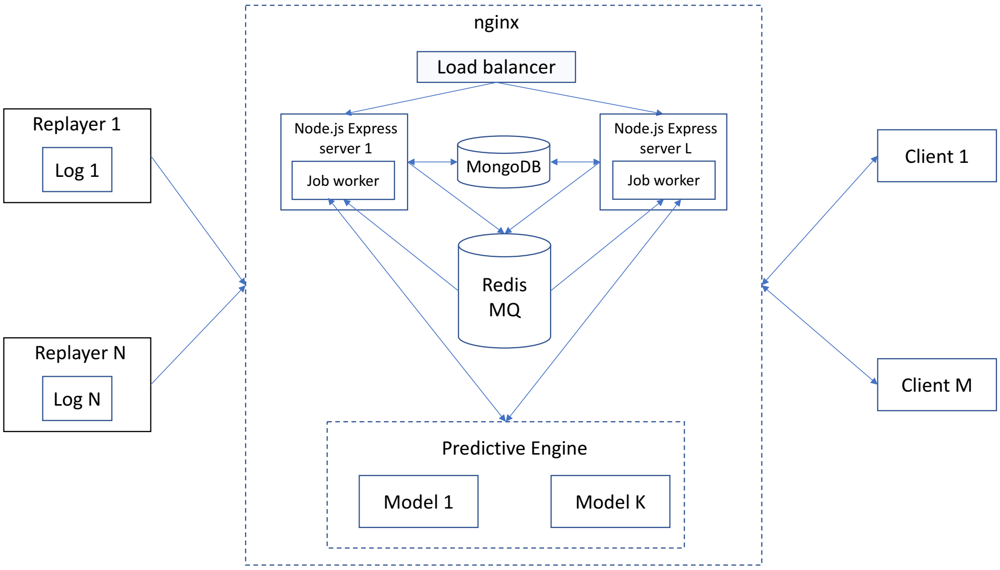


Figure 8. Developed system architecture

requires the following fields for Event:

- case and log identification values
- activity name
- when it has been occurred
- whether event is the last in a case or not
- event number within a case

The value of an event number within a case can be replaced by information from analyzing timestamps of the message.

Case stores fields concerning both descriptive and predictive values. When the case is finished, all fields hold actual values since they are already known at this point. The descriptive fields include following:

- case and log identification values
- whether case is finished or not
- start time
- latest event time
- current trace length and prefix

Completion time and duration are known when a case is finished. Before that, however, those fields are calculated based upon remaining time predictions. All other fields depend on available predictive models. One should note that the names of such fields can be arbitrary (as configured), but they must be present. If any of the earlier mentioned required fields are missing, Mongoose module would not store such information. Otherwise, the system is likely to malfunction given the faulty messages.

When a message with an event reaches a Node.js server instance, it is firstly stored in MongoDB. If the event in question is the first event of a case, then it creates a document for a case based on information from the received message; otherwise, it merely updates a corresponding case. Then, all relevant clients are updated. Eventually, the program requests a module for calculations to apply all predictive models, stores the results in the database, and updates all relevant clients again. To avoid inconsistent information in a dashboard, the results from all available predictive models are sent together. If the event is the last one in a case, then the system acts the same way, except for its interaction with the predictive engine since the final state of the case is already known and there is hence no need to apply models.

To reduce the load from the predictive engine, there has been used the load shedding technique, which implies skipping unnecessary actions in stream processing. Thus, the developed system skips prediction calculations for a given partial trace, if a newer event of that case arrives.

Two types of predictive models in the developed software have been used: Remaining cycle time and outcome predictions. The number of different outcomes can be arbitrary. All the required information for a correct performance of the predictive engine is specified in the configuration file. The example of the configuration file can be seen in Appendix I.

To simplify testing and development, two additional modules have been written. The first one is responsible for database creation (or recreation if such already exists). The second one is responsible for a message queue setup. While developing an application, one should note that the system should be in the initial state on every execution. That is why previously mentioned modules are run automatically when the application is started in the development mode through a NODE_ENV environmental variable.

In order to avoid problems with the setup of different versions of MongoDB and Redis on various operation systems, there have been used Docker containers from Docker Hub registry[1]. For better deployment management of an application to a remote server, pm2[2] tool has been used. It is an advanced process manager for Node.js which allows an application's state managing. Due to the fact that it runs in background, it is possible to manage different applications at the same time.

Demo of the developed project has been deployed to a remote server, kindly provided by the University of Tartu. Ubuntu 16.04.2 has been installed on the hosted machine. It has 8 GB of RAM, 20 GB of HDD and Intel i7 quad-core processor. The

[1] https://hub.docker.com

[2] http://pm2.keymetrics.io/

process of deployment has been implemented based on [8]. Nginx configuration file and script for application deployment can be seen in Appendix II and Appendix III correspondingly.

Current implementation successfully serves a role of a prototype for a predictive process monitoring engine. However, there exists a further plan for its improvement. Dockerization of the system, which means putting all system components into a docker container, would prevent possible problems with system setup. Another improvement includes test coverage of the project. When a system is growing or when several developers work on the same software project, it increases the risk of breaking changes in the code. Integration tests (testing of interaction between different components such as database or message queue) and unit tests (testing of functions functionality) enhance collaborative development for complex systems. Finally, there is a plan to implement components for connecting a system to real-time event streams from external sources.

A more detailed technical documentation is available at our GitHub repository (Appendix IV).

4 Evaluation

In this section, we aim to evaluate the developed software solution as to its practicality in real-life applications. We start by describing the evaluation framework. Next, we show how the predictive models are built and how their accuracy is measured. Then we validate the developed solution from a user's perspective with respect to its original requirements outlined in Section 3.1. Finally, we perform a runtime performance analysis of our solution.

4.1 Context and Motivation

As a validation scenario, we consider the loan and overdraft approvals process are originating from a bank in the Netherlands. The anonymized event log of the corresponding business process was first published as a part of the 2012 Business Process Intelligence Challenge [30] and later substantially extended as the new log for the 2017 Business Process Intelligence Challenge [31]. The first log consists of a snapshot of 13 087 loan application processes processed by the bank over a period from October 1, 2011 to March 15, 2012, while the other one contains 31 509 cases spanned for the applications filed between January 1, 2016 and January 1, 2017. In the rest of the paper, we will refer to these logs as *BPIC'2012* and *BPIC'2017* respectively.

Both logs refer to the same application process. It starts when a client lodges an online application form. After a few automated checks, a bank specialist contacts the applicant regarding the additional information. This information is usually obtained via phone calls. If an applicant is deemed to be eligible, an offer is sent to him by mail. A customer considers the offer and in case of acceptance sends it back to the bank for further evaluation. In case it is incomplete, missing information is added by reaching the customer again. Then a final application assessment is performed, after which the loan is approved and issued. If the customer does not qualify for the loan, the application is rejected.

The events from the logs are classified into three types, each of which can be regarded as separate subprocess [30, 31]:

- Application Events – refers to states of the application itself. Event names of this type start with "A_" prefix.
- Offer Events – refers to states of an offer communicated to the customer. Events start with "O_" prefix.
- Work item Events – refers to states of work items that occur during the approval process. Events of this type record bank's internal procedures performed for the loan approval process. Their names start with "W_" prefix.

Like any other financial institution, the bank at question has strict process performance objectives specified as internal policies. One such objective stipulates that an application should be decided within a specified amount of time. The rationale is that overly long application processing times cause customer dissatisfaction, and eventually, customer churn.

Furthermore, such deadlines violations erode the bank's potential profit as fewer customers can be served. From a customer's perspective, the time prediction also adds value.

For example, when a customer phones the bank for information about the submitted application, they can be given an estimate for the remaining processing time. Besides that, the bank aims to maximize the amount of customers accepting the loan offer and the total amount of issued loans.

In a real setting, these performance targets are not always fulfilled, for various reasons. One way to steer the process execution to keep it within the performance bounds is to allocate the organization's resources in such a way that cases that are expected to deviate from the prescribed behavior receive a higher priority. In this way, the potential problems can be proactively solved, or at least mitigated, before they actually occur.

Having a real-time process monitoring engine that keeps track of a large amount of running cases and makes predictions with respect to their future state is highly beneficial, as the process workers will receive a continuous guidance towards the desired direction of the process.

Having said that, two important variables that the bank in question would be interested in predicting are: (i) the remaining processing time, and hence total case duration and (ii) whether or not the customer will accept the loan offer. In the remainder of the section, we show how predictive models for these variables can be built and evaluated, as well as how the developed predictive monitoring engine functions from a user's perspective, in line with the established requirements.

4.2 Construction of Predictive Models

In this subsection, we first describe the preparation of necessary datasets from the original event logs. Then we present a model training procedure. Finally, we thoroughly assess the models with respect to the chosen evaluation metrics.

4.2.1 Data preparation

To make the predictions, we need to first train the predictive models. Two things have to be kept in mind at this stage:

- A standard practice in machine learning is to use 70-80% of the original dataset to train the models, and the rest to test it. The two sets have to be completely disjoint [4].

- In a real-life setting, the data about the historical process execution is used to train a model that is later tested on a *future* data.

To satisfy these constraints, we propose to split the data along the temporal dimension as follows. We fix a time point in the process execution that serves a “present” moment, so that the cases that have been completed prior to this point become a training set, and the cases starting after this point join a test set.

Specifically, for the BPIC’2017 log, we picked October 1, 2016 as a current time point. Thus, cases that are *entirely* contained in the interval from January 1, 2016 to October 1, 2016 form a training set, and cases between October 1, 2016 and January 1, 2017 form a test set. It should be noted that cases that were running as of October 1, 2016 midnight are not included in either set. However, for the BPIC’2012 log that already has a lower number of cases, it would mean that the total number of cases available for training and test would be rather low. Furthermore, BPIC’2012 log has fewer data attributes. Therefore, for this log, we did not mandate the two sets to be completely separated, in a temporal dimension. Specifically, we ordered the cases in that log by their start time and used the first 70% for training and the rest for the test.

Furthermore, the BPIC’2017 log we consider events of all three types, while for the BPIC’2012 log we only consider the work item subprocess ("W_").

4.2.2 Model training

As mentioned in subsection 4.1, two variables that would be beneficial for a bank to predict are a case duration and whether or not a customer will accept a loan offer. The latter can be modeled as a boolean variable, and predicted via a classification model. As for a case duration, we experiment with the two alternatives:

- Predict whether a case duration will fulfill a specified threshold objective. If no such objective is provided by the bank, one can use a mean or median case duration, or some other distinctive value. In this case, we would need a classification model.
- Predict the numerical value of the remaining processing time via the regression model. Consequently, we can calculate the case duration by summing up the elapsed processing time with the obtained prediction.

For the BPIC’2012 log, as the work item subprocess does not contain the loan acceptance data, we only considered two time-based models.

To encode the process execution traces as feature vectors, we used the index-based encoding proposed by Leontjeva et al. [13]. This type of encoding has been shown to achieve a relatively good accuracy and stability when making early predictions of the process outcome [13, 27, 33]. As a reference, we used the Python implementation of the predictive monitoring provided by Teinemaa et al.[1].

In the index-based encoding, along with the static case attributes, one feature is generated for each attribute of each executed event. In general, for a case i with U case attributes $\{s_1, \ldots, s_U\}$ containing M events $\{e_1, \ldots, e_M\}$, each of them having an associated set of attributes $\{d_1^1, \ldots, d_1^R\}, \ldots \{d_M^1, \ldots, d_M^R\}$ of length R, the resulting feature vector would be [33]:

$$\vec{X}_i = (s_1, \ldots, s_U; d_1^1, \ldots, d_1^R, \ldots d_M^1, \ldots, d_M^R) \tag{3}$$

Thus, the length of the feature vector would be $U + M \cdot R$. Because the length of the feature vector obtained via index-based encoding depends on the number of so far executed events, a separate model has to be trained for each length of a partial trace.

In our experiments, we fit a random forest and a gradient boosted tree model (GBM), both for classification and regression. To select an optimal value for the training parameters, we performed a grid search over a set of two important parameters:

- For random forest – number of trees and the number of features for random sampling
- For GBM – number of trees and learning rate

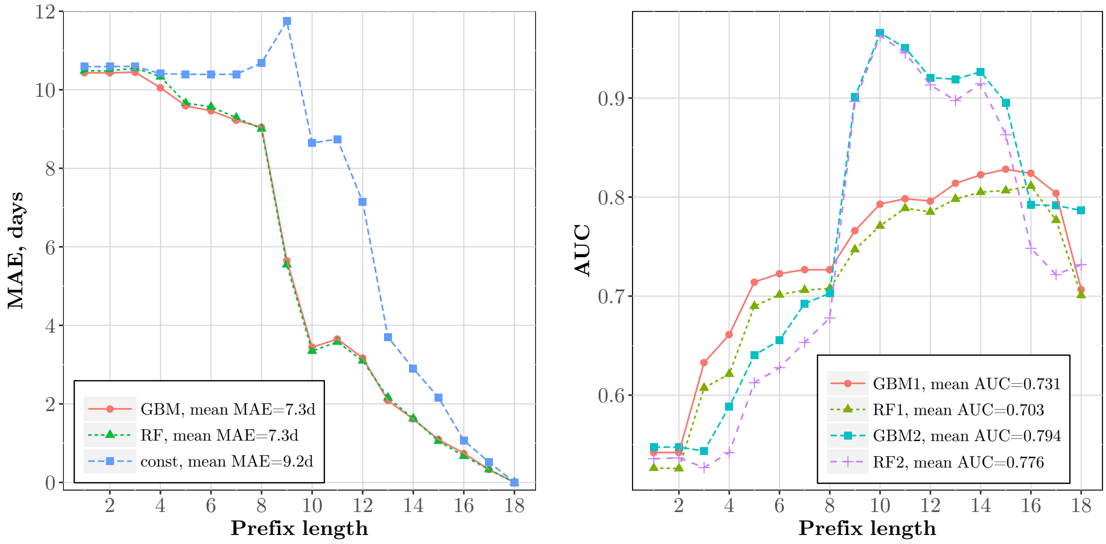
For each pair of parameters, we train a model and evaluate its performance using five-fold cross validation. Finally, we choose the pair that achieves the highest performance and use it to evaluate the models.

4.2.3 Model evaluation

To evaluate the performance of the models, we use Mean Absolute Error (MAE) for regression tasks and Area Under Curve (AUC) for the classification tasks.

In Figures 9a and 10a we show the remaining time prediction accuracy for the BPIC'2012 and BPIC'2017 respectively. As a baseline predictor for regression task, we compared with a simple *constant* model that always outputs the mean remaining value after a given number of events have elapsed [1]. As expected, the predictive models outperform the constant regressor across all possible prefix lengths, with GBM and random forest achieving nearly the same accuracy. Another interesting observation is that in the beginning, with very short cases, the accuracy of predictive models is not much better than that of a mean model. This is due to the lack of useful data attributes when a case just starts. When a case progresses, more data become available, thus enabling more accurate predictions. With longer prefixes, the error becomes negligible, mostly because the actual remaining time also decreases. Furthermore, when measuring

[1] http://github.com/irhete/PredictiveMonitoringWithText



(a) Remaining time

(b) Case outcomes

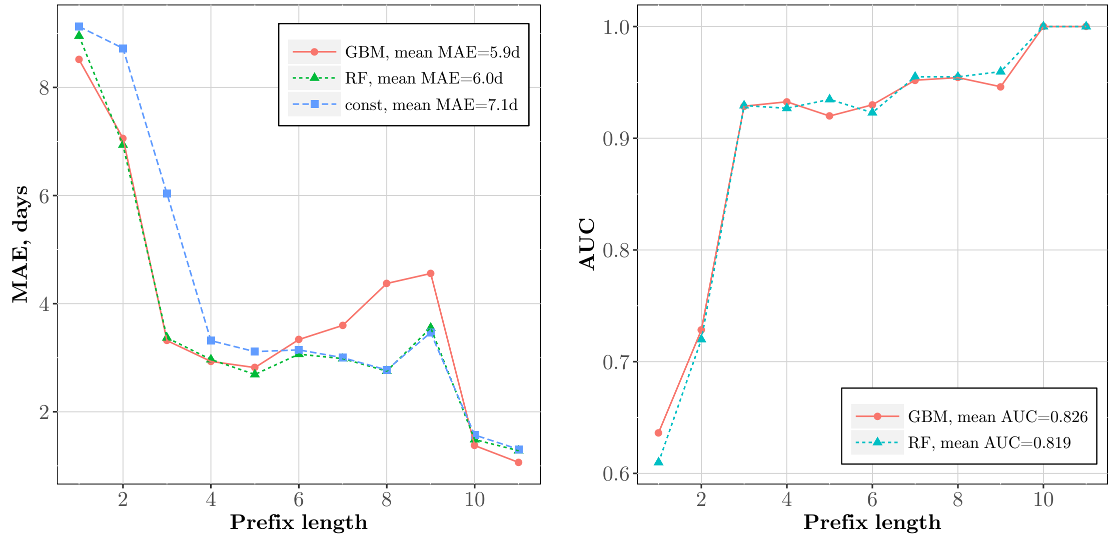
Figure 9. Prediction accuracy for the BPIC'2017 log across different prefix lengths

error for longer prefixes, we have fewer test samples that ever reach that length. Thus, error measurements become less reliable.

Figure 9b shows an accuracy of the classification models. The first model, marked as GBM1 and RF1, was trained to predict whether a customer will accept the loan offer. The other model, marked as GBM2 and RF2, predicts whether a case duration will be within 30 days, which is a mean case duration for this event log. Similarly, AUC improves with longer prefixes. However, for the very long prefixes, it starts to slightly decrease, probably since only "difficult" cases reach that length. Figure 10b shows an accuracy for the classification model that predicts whether a case duration will be within a required threshold, that is 16 days. Here, we do not see the signs of performance decay with higher lengths.

Finally, we assess the usefulness of the models over time. As the model is used on newer data, it may experience performance decay in the presence of the concept drift [2]. Thus, it needs to be retrained occasionally. As more and more cases complete over time, they should be included in a new training set.

We analyze the model decay using only BPIC'2017 log, as BPIC'2012 does not cover a sufficient time frame for this kind of analysis. We split the test set into three parts: (i) cases that started in October 2016; (ii) cases started in November; (iii) cases started in December. Since the case arrival rate is not constant, the first part contains 46% of the original test set, the second one 42%, and the last one 12%. For our experiments, we perform the analysis only for the GBM model, as it slightly outperforms random forest. Furthermore, we do not consider only the classification model that pre-



(a) Remaining time

(b) Case outcome

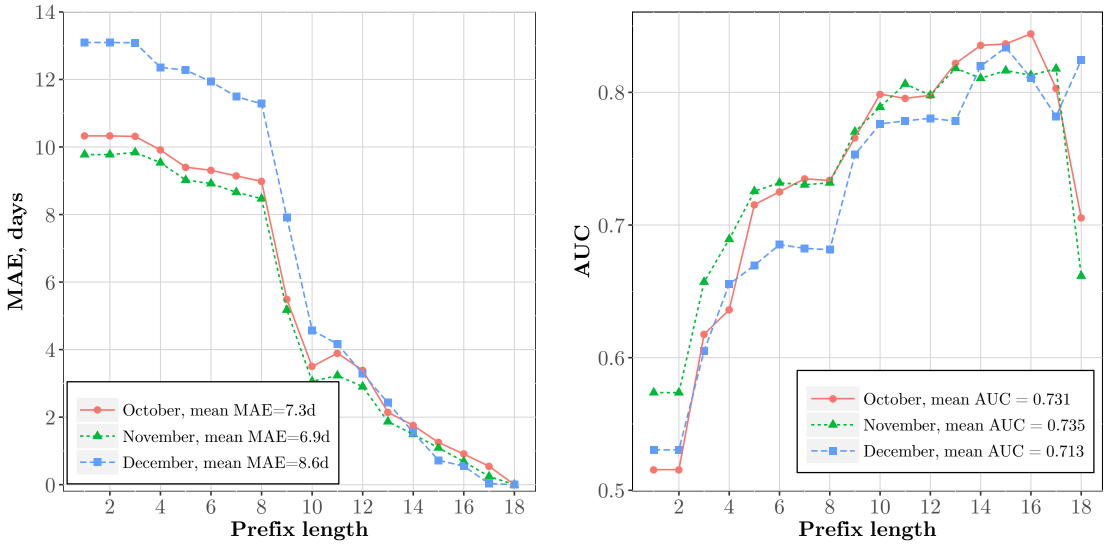
Figure 10. Prediction accuracy for the BPIC'2012 log across different prefix lengths

dicts the loan acceptance.

Figure 11 illustrates our findings. Surprisingly, the classification and regression model trained on the January to September data is slightly more accurate for cases started in November than for the October cases. However, for the December cases, one can see the signs of performance decay.

4.3 User Validation

In this section, we describe the implemented engine as perceived by a potential user, though the user interface. Figure 12 describes initial view which user faced when system is loaded. The main part of it is a Detail view which is represented in a form of a table. It lists both currently *ongoing* cases (which have gray rows) as well as already *completed* ones (which have green rows). For all process cases, we display descriptive statistics such as the number of elapsed events, the list of elapsed events in a given case (this field is hidden by default), case start time and the time of the latest event completion. For ongoing cases, additionally we estimate the *predicted* completion time, *predicted* case duration, the probability of a case to be "slow", i.e to take more time than a certain threshold and the probability of a case to be rejected, meaning that a customer will not accept the loan offer (only relevant for the BPIC'2017). For completed cases, instead, we show the *actual* completion time, *actual* case duration, and the *actual* case outcome, i.e whether the case has been "slow" and whether it has been rejected. The table allows sorting cases on column basis and managing which columns are visible. Moreover,



(a) Remaining time (b) Case outcome

Figure 11. GBM model performance decay over the 3 months for the BPIC'2017 log

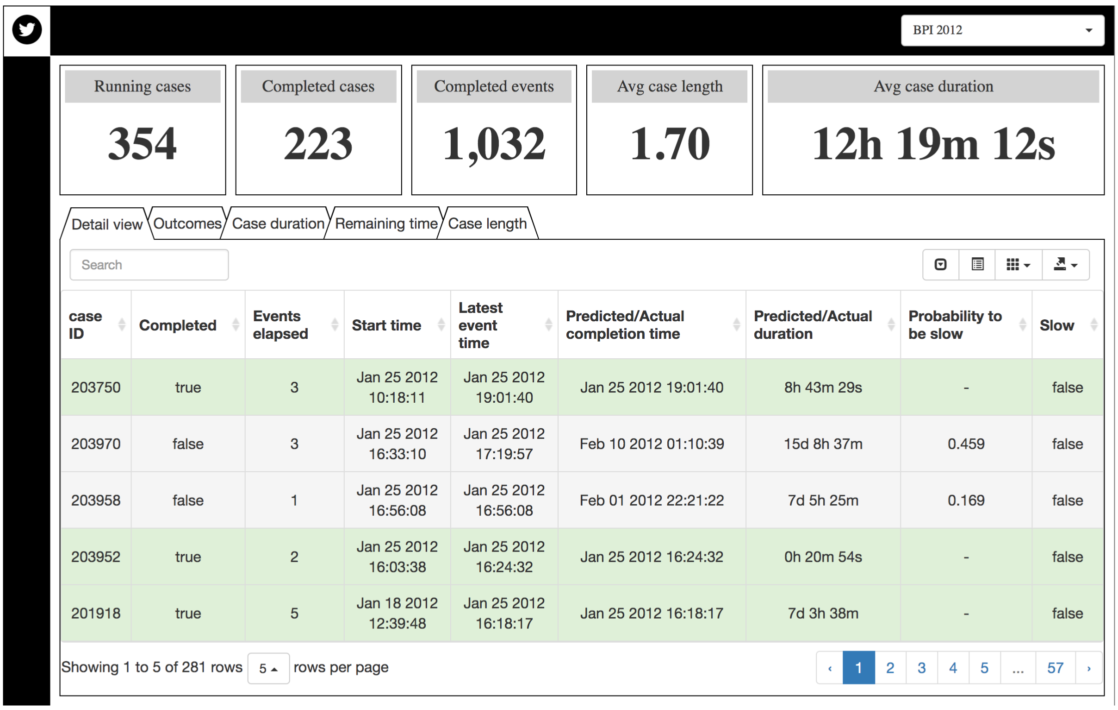


Figure 12. Main application view

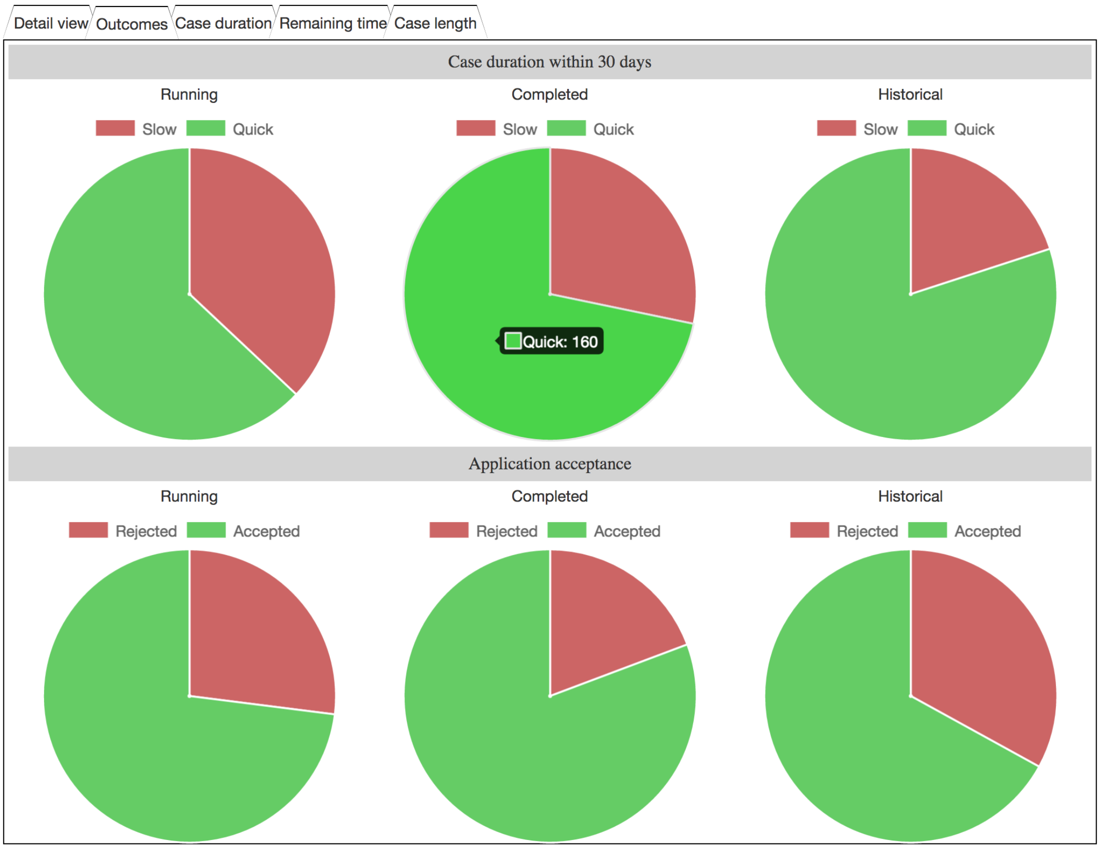


Figure 13. Outcomes tab

pagination mechanism enables smooth navigation between table rows. Besides that, color-coding is used for cells as well in addition to rows. So, if the case has the unwanted outcome (either predicted or actual), then the corresponding cell has a red color. Thus, functional requirements FR1, FR2, FR3, and FR4 are validated.

Additionally, Detail view contains search box for filtering the results, pagination support, and toolbar section. Available options in the toolbar allow users to toggle pagination and table/list view, as well as to choose which columns to display and to export the current table to a JSON, XML, CSV, etc. format. The latter can be used to conveniently produce tailor-made statistics and visualizations not currently offered in the tool. Thus, functional requirements FR5 and FR6 are validated.

On the upper part for all views, there is an Aggregated process state panel. It includes the number of running and so far completed cases, the number of occurred events, average number of events per case and average case duration. Thus, functional requirement FR7 is validated.

Figure 13 shows the Outcomes tab which visualizes binary case outcomes for three

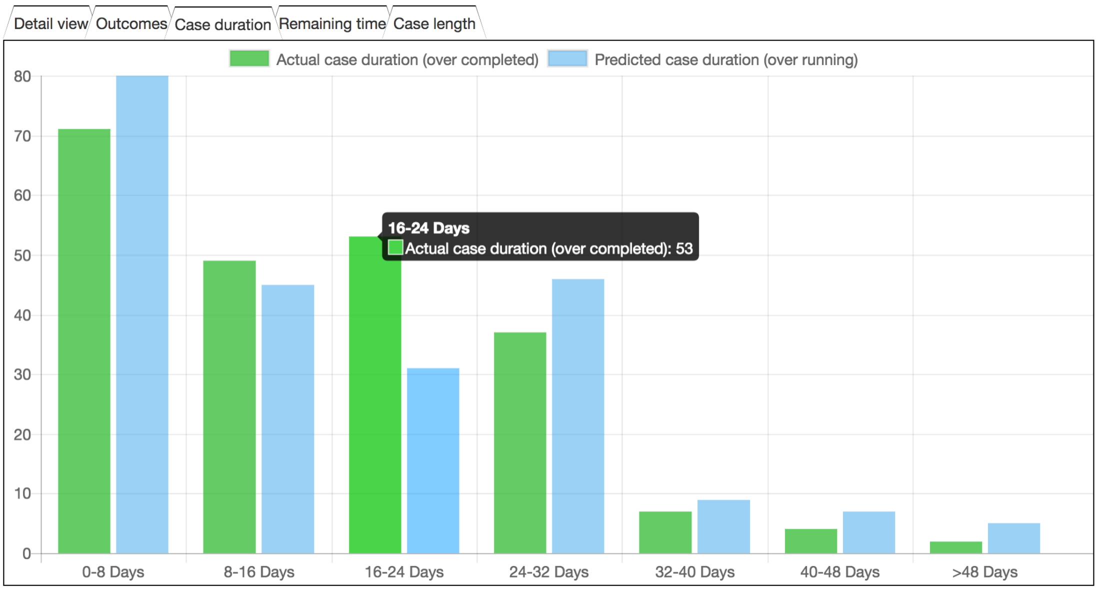


Figure 14. Distribution of case duration tab

types of cases: ongoing (outcome is predicted), completed (outcome is actual) and historical (over the dataset that was used to train the predictive models). As the models are currently only retrained once, the latter diagram is static. The Case duration tab which is depicted on figure 14 shows a bar chart histogram of case duration distribution among ongoing and completed cases. The Remaining time tab shows a histogram of remaining cycle time for ongoing process cases. Similarly, on the Case length tab, one will find case lengths in terms of the number of events. Thus, functional requirement FR8 is validated.

In top right corner there is a possibility to choose which business process to observe. In the current demo there are processes based on BPIC'2012 and BPIC'2017. After user switches to another process, all information in a dashboard updated automatically. At the same time, every user has a possibility to choose which process to observe without interfering with others. Thus, functional requirement FR9 is validated.

5 Conclusions and Future Work

5.1 Conclusion

Although process monitoring tools successfully exist on the market for quite a long time, *predictive* process monitoring tools have not appeared yet. At the same time, academic society is working on techniques and methods for predictive process monitoring and continuously delivers new findings. Within a scope of the current thesis project, the gap between software engineering and academic research domains has been fulfilled.

As a result of this thesis, there has been designed and developed a prototype for predictive process monitoring engine in the form of a configurable full-stack web framework. Nirdizati is a dashboard-based monitoring tool, which is updated based on an incoming stream of events. However, unlike classical monitoring dashboards, Nirdizati does not only focus on the current state of business process executions but also, and more importantly, it sheds light on their future state. Also, the developed software allows handling event streams that originate from multiple business processes. Thus, users can switch between business processes depending on their responsibility areas. Dashboard representation, which has multiple visualization options based on device's screen size, includes the results of the predictions, as well as the real-time summary statistics about the process execution.

The implemented software project includes predictive models for outcome predictions and remaining cycle time; however, an even more valuable achievement is a possibility for easy system extension by new predictive models which, in turn, can be written in any programming language. As the project has been released as open source software under the GNU Lesser General Public License, all data scientists, who work in this area, can contribute to the predictive engine by providing new models. On the other hand, software engineers can work on developing new features and improving the current state of the project.

It is not only academic society who can benefit from the developed solution. Enterprise organizations can already start integrating the proposed software and successfully apply it for business process predictive monitoring. Process workers and operational managers – typical users of such systems – are hence capable of making more informed, data-driven decisions to get a better control of business process execution direction. "How long will it take to handle this claim? Will this loan offer be accepted?" are the standard questions that Nirdizati aims to answer.

The solution has been successfully validated on the established requirements using event streams corresponding to two real-life business processes. In a test case scenario, we showed how the developed predictive monitoring engine could be leveraged in an organization. However, in practice, this type of solution needs a deep integration with existing enterprise systems. Furthermore, the adoption of such a solution would affect the business processes in the company, so it comes with process change.

5.2 Future work

There is a clear vision regarding further improvement of developed software. The priority is an improvement of stream management mechanism via Apache Kafka streaming platform integration into the project. It would facilitate an integration of Nirdizati into enterprise systems.

The next priority is to enhance user interaction with the dashboard tool. For that, user authentication and authorization mechanism are required. It supposes a development of private cabinets where the user would be able to observe their business process. As the first step, those processes can be set by us; however, the plan is to give a possibility for the end users to upload their log or to specify stream events source. After that, the user should be able to configure business process execution straight from the account settings.

In addition to matters above, there is a plan to improve customizable analytics for the end user. This includes a possibility to observe the results within a set time interval. Another improvement would allow setting custom intervals for a bar chart width. Also, there is a need for the feature of adding additional filter options both to Detail view and to any of the charts views. In Detail view, filters can run on a column basis, while on any chart view user can filter the chosen cases by clicking on a particular bar or pie chart section.

Last but not least there is an idea to implement a *prescriptive* monitoring concept. While predictive monitoring exploits historical process execution data to determine the probable future course of an ongoing process case, prescriptive monitoring takes a step further by also recommending actions to get benefits from the predictions. Prescriptive process monitoring goes far beyond the scope of the current master's thesis and is one of the conceptual directions which tool is planning to follow.

References

[1] B. Abraham and J. Ledolter. *Statistical Methods for Forecasting*. Wiley Series in Probability and Statistics. Wiley, 2009.

[2] Charu C Aggarwal. *Data Mining - The Textbook*. Springer, 2015.

[3] Apache. Introduction to Apache Kafka. `https://kafka.apache.org/intro#`. Accessed: 2017-05-16.

[4] Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

[5] Dominic Breuker, Martin Matzner, Patrick Delfmann, and Jörg Becker. Comprehensible predictive models for business processes. *MIS Quarterly*, 40(4):1009–1034, 2016.

[6] Raffaele Conforti, Massimiliano de Leoni, Marcello La Rosa, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. A recommendation system for predicting risks across multiple business process instances. *Decision Support Systems*, 69:1–19, 2015.

[7] Massimiliano de Leoni, Wil M. P. van der Aalst, and Marcus Dees. A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems*, 56:235–257, 2016.

[8] DigitalOcean. How To Set Up a Node.js Application for Production on Ubuntu 16.04. `https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-16-04` Accessed: 2017-05-16.

[9] Marlon Dumas, Marcello La Rosa, J Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer Berlin Heidelberg, 2013.

[10] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.

[11] João Gama. *Knowledge Discovery from Data Streams*. Chapman and Hall / CRC Data Mining and Knowledge Discovery Series. CRC Press, 2010.

[12] Geetika T Lakshmanan, Davood Shamsi, Yurdaer N Doganata, Merve Unuvar, and Rania Khalaf. A markov prediction model for data-driven semi-structured business processes. *Knowledge and Information Systems*, 42(1):97–126, 2015.

[13] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria Maggi. Complex Symbolic Sequence Encodings for Predictive Monitoring of Business Processes. In *Business Process Management - 13th International Conference*, pages 297–313, 2015.

[14] Fabrizio Maria Maggi, Chiara Di Francescomarino, Marlon Dumas, and Chiara Ghidini. Predictive monitoring of business processes. In *Advanced Information Systems Engineering - 26th International Conference, CAiSE*, pages 457–472. Springer, 2014.

[15] A. Mardan. *Full Stack JavaScript: Learn Backbone.js, Node.js and MongoDB*. Apress, 2015.

[16] Andreas Metzger, Rod Franklin, and Yagil Engel. Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case. In *2012 Annual SRII Global Conference*, pages 313–322, 2012.

[17] Andreas Metzger, Philipp Leitner, Dragan Ivanovic, Eric Schmieders, Rod Franklin, Manuel Carro, Schahram Dustdar, and Klaus Pohl. Comparing and combining predictive business process monitoring techniques. *IEEE Trans. Systems, Man, and Cybernetics: Systems*, 45(2):276–290, 2015.

[18] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[19] A. Mouat. *Using Docker: Developing and Deploying Software with Containers*. O'Reilly Media, 2015.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] Mitch Rees-Jones, Matthew Martin, and Tim Menzies. Better predictors for issue lifetime. *CoRR*, abs/1702.07735, 2017.

[22] Andreas Rogge-Solti and Mathias Weske. Prediction of business process durations using non-markovian stochastic petri nets. *Information Systems*, 54:1–14, 2015.

[23] M. Satheesh, B.J. D'mello, and J. Krol. *Web Development with MongoDB and NodeJS*. Packt Publishing, 2015.

[24] Arik Senderovich, Matthias Weidlich, Avigdor Gal, and Avishai Mandelbaum. Queue mining - predicting delays in service processes. In *Advanced Information Systems Engineering - 26th International Conference, CAiSE*, pages 42–57, 2014.

[25] J R Spiegel, M T McKenna, G S Lakshmanan, and P G Nordstrom. Amazon US Patent Anticipatory Shipping. *Amazon Technologies Inc*, 12.

[26] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. Predictive business process monitoring with LSTM neural networks. In *Advanced Information Systems Engineering - 29th International Conference, CAiSE*, page To appear. Springer, 2017.

[27] Irene Teinemaa, Marlon Dumas, Fabrizio Maria Maggi, and Chiara Di Francescomarino. Predictive business process monitoring with structured and unstructured data. In *Business Process Management - 14th International Conference, BPM*, pages 401–417, 2016.

[28] Wil M. P. van der Aalst, M. H. Schonenberg, and Minseok Song. Time prediction based on process mining. *Information Systems*, 36(2):450–475, 2011.

[29] Sjoerd van der Spoel, Maurice van Keulen, and Chintan Amrit. Process prediction in noisy data sets: a case study in a dutch hospital. In *International Symposium on Data-Driven Process Discovery and Analysis*, pages 60–83. Springer, 2012.

[30] B.F. van Dongen. BPI challenge 2012. `https://www.win.tue.nl/bpi/doku.php?id=2012:challenge`. Accessed: 2017-05-15.

[31] B.F. van Dongen. BPI challenge 2017. `https://www.win.tue.nl/bpi/doku.php?id=2017:challenge`. Accessed: 2017-05-15.

[32] Ilya Verenich. A general framework for predictive business process monitoring. In *Proceedings of CAiSE 2016 Doctoral Consortium co-located with 28th International Conference on Advanced Information Systems Engineering*, 2016.

[33] Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Chiara Di Francescomarino. Minimizing overprocessing waste in business processes via predictive activity ordering. In *Advanced Information Systems Engineering - 28th International Conference, CAiSE*, pages 186–202, 2016.

[34] Ilya Verenich, Hoang Nguyen, Marcello La Rosa, and Marlon Dumas. White-box prediction of process performance indicators via flow analysis. In *ICSSP*, page To appear. ACM, 2017.

[35] Mark von Rosing, Henrik von Scheel, and August-Wilhelm Scheer, editors. *The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM, Volume I.* Morgan Kaufmann/Elsevier, 2015.

[36] Wikipedia. Reverse proxy. `https://en.wikipedia.org/wiki/Reverse_proxy`. Accessed: 2017-05-16.

Appendix

I. Application configuration file

```
{
  // app specific settings
  "app": {
    "port": 8080
  },

  "mongoose": {
    "uri": "mongodb://localhost:27017/dev",
    "options": {}
  },

  "redis": {
    "connection": {
      "host": "localhost",
      "port": 6379,
      "ns":  "vis-pred"
    },
    "jobQueue": "jobs-queue"
  },

  // default config for replayer
  "replayer": {
    "log": "bpi_17",
    "request": {
      "hostname": "localhost",
      "port": 8080,
      "path": "/event",
      "method": "POST",
      "headers": {
        "Content-Type": "application/json",
        "Authorization": "replayer:12345"
      }
    },
    "accelerator": 20,
    "isTestMode": false,
    "testInterval": 5000
  },

  // BPI 2017 configurations
  "bpi_17": {
    "path": "data/test_bpi17_sorted.csv",
    "timeField": "time",
    "timeFormat": "YYYY-MM-DD HH:mm:ss",
```

```
  "caseIdField": "sequence_nr",
  "eventNameField": "activity_name",

  "ui": {
    "daysInterval": 8,
    "barsCountForTimeIntervals": 7,
    "barsCountInLengthDistribution": 9,
    "barsWidthForLength": 3
  },

  "methods": {
    "outcomes": {
      "slow_probability": {
        "wd": "/PredictiveMethods/CaseOutcome/",
        "executable": "python",
        "args": ["test.py", "", "bpi17", "label2"],
        "type": "temporal",
        "probabilityThreshold": 0.52,
        "property": "slow",
        "criterion": "duration",
        "criterionThreshold": 2592000000,
        "ui": {
          "name": "Case duration within 30 days",
          "labels": ["Slow", "Quick"],
          "historical": [1390, 1398]
        }
      },
      "rejected_probability": {
        "wd": "/PredictiveMethods/CaseOutcome/",
        "executable": "python",
        "args": ["test.py", "", "bpi17", "label"],
        "type": "logical",
        "probabilityThreshold": 0.45,
        "property": "rejected",
        "label": "label",
        "ui": {
          "name": "Application acceptance",
          "labels": ["Rejected", "Accepted"],
          "historical": [877, 1911]
        }
      }
    },
    "remainingTime": {
      "wd": "/PredictiveMethods/RemainingTime/",
      "executable": "python",
      "args": ["test.py", "", "bpi17"]
    }
  },

```

```
    "replayer": {
      "request": {
        "hostname": "localhost",
        "port": 8080,
        "path": "/event",
        "method": "POST",
        "headers": {
          "Content-Type": "application/json",
          "Authorization": "replayer:12345"
        }
      },
      "accelerator": 50,
      "isTestMode": true,
      "testInterval": 6000
    }
  }
}
```

II. Nginx configuration file

```
upstream nirdizati {
  least_conn;
  server localhost:8081;
  server localhost:8082;
  server localhost:8083;
  server localhost:8084;
  server localhost:8085;
}

server {
  listen 80;
  server_name nirdizati.cs.ut.ee;
  gzip on;
  gzip_comp_level 5;
  gzip_min_length 256;
  gzip_proxied any;
  gzip_vary on;

  location / {
    proxy_pass http://localhost:8080;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
  }

  location /event {
    proxy_pass http://nirdizati;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
  }
}
```

III. Deployment script using process manager

```
export NODE_ENV=development
export NODE_PATH=.

sudo npm --loglevel=error install
cd src
sudo npm --loglevel=error install
gulp prod
cd ..

docker volume rm $(docker volume ls -qf dangling=true)
docker rm -f some-mongo some-redis
docker run --name some-mongo -d -p 27017:27017 mongo
docker run --name some-redis -d -p 6379:6379 redis redis-server --appendonly yes

sudo pm2 delete all

sudo pm2 start -f server.js -- 8080
sudo pm2 start -f server.js -- 8081
sudo pm2 start -f server.js -- 8082
sudo pm2 start -f server.js -- 8083
sudo pm2 start -f server.js -- 8084
sudo pm2 start -f server.js -- 8085

sudo pm2 start -f libs/replayer.js -- bpi_12
sudo pm2 start -f libs/replayer.js -- bpi_17
```

IV. Source code

The source code has been released as open source software under the Lesser GNU Public License (L-GPL) at `http://github.com/nirdizati`. The developed engine has been deployed at the server belonging to the Institute of Computer Science and can be accessed at `http://nirdizati.cs.ut.ee`.

V. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Andrii Rozumnyi**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

   of my thesis

   A Dashboard-based Predictive Process Monitoring Engine

   supervised by Ilya Verenich

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 18.05.2017