UNIVERSITY OF TARTU

Institute of Computer Science

Software Engineering Curriculum

Aladdin Shikhizada

A Tool for Prescriptive Monitoring of Business Processes

Master's Thesis (30 ECTS)

Supervisor: Fabrizio Maria Maggi

Tartu 2020

A tool for prescriptive monitoring of business processes

Abstract: Most companies are deeply concerned about monitoring their business processes. It has special importance for them because in this way they can detect and avoid undesired outcome that can happen. In this context, prescriptive process monitoring techniques aim at analyzing historical behavior of business processes recorded in event logs using machine learning algorithms and then use this information for providing recommendations about actions to take in ongoing process executions to achieve a desired outcome.

In this research, our goal is to build a process-oriented recommender system by implementing two approaches for prescriptive process monitoring. The system takes an event log as input, builds a predictor based on the information retrieved from the log and use it for providing recommendations on validation data.

Keywords: Predictive Process Monitoring, Prescriptive Process Monitoring, Process Mining, Declarative process models, Process discovery, Conformance checking, Recommender systems

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

Tööriist äriprotsesside ettekirjutatavaks jälgimiseks

Lühikokkuvõte: Enamik ettevõtteid peab oma äriprotsesside jälgimist eriti oluliseks. See omab ettevõtete jaoks erilist tähtsust, kuna võimaldab neil tuvastada ja vältida soovimatuid tagajärgi, mis võivad protsesside tulemusena tekkida. Selles kontekstis on ettekirjutatavate protsesside jälgimise meetodite eesmärk analüüsida sündmuselogidesse salvestatud äriprotsesside ajaloolist käitumist masinõppe algoritmide abil ja seejärel kasutada saadud teavet soovituste tegemiseks toimingute kohta, mida tuleb käimasolevate protsesside täitmisel soovitud tulemuse saavutamiseks teha.

Selles uurimistöös on meie eesmärk üles ehitada protsessidele orienteeritud soovitussüsteem, rakendades kahte lähenemisviisi ettekirjutatava protsessi jälgimiseks. Süsteem võtab sisendina sündmuselogi, loob logist kogutud teabe põhjal ennustaja ja kasutab seda andmete valideerimiseks ja soovituste andmiseks.

Võtmesõnad: ennustav protsesside jälgimine, ettekirjutatav protsesside jälgimine, protsessikaeve, deklaratiivsed protsessimudelid, protsesside väljaselgitamine, vastavuskontroll, soovitussüsteemid

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

List of Tables

List of Figures

List of Abbreviations

MXML	Mining Extensible Markup Language
XES	Extensible Event Stream
CSV	Comma-Separated Values
ETL	Extract, Transform, Load
IEEE	Institute of Electrical and Electronics Engineers
MP-Declare	Multi-Perspective Declare
RF	Random Forest
LGBM	Gradient Boosting
UI	User Interface
REST	Representational State Transfer
JSON	JavaScript Object Notation
AJAX	Asynchronous JavaScript and XML
PM4Py	Process Mining for Python
ERP	Enterprise Resource Planning
PAR	Process Aware Recommender Systems
UWV	Employee Insurance Agency (Netherlands)

1. Introduction

In recent years business processes are becoming more and more complex which makes managing, monitoring and tracking them more difficult. Monitoring business processes is a crucial activity in companies. Any problem that can happen during the execution of business processes may lead to the extra financial expenses for the organization as well as losing customers due to their dissatisfaction. In order to avoid such issues, managers use business process monitoring tools. Generally, the most common used tools are based on reactive compliance monitoring in which violations can be detected only after they have happened [1]. In contrast, predictive process monitoring allows users to predict an undesired outcome before the violation occurs so that actions can be taken to prevent it. In this research, we go one step further and implement two methods for monitoring business processes prescriptively. These methods not only predict the occurrence of a future undesired outcome but also suggest what to do to prevent the undesired outcome to occur. For example, in the medical field there are some diseases that need to be predicted as early as possible. By applying prescriptive process monitoring, we can inform the patients at an early stage of the illness and recommend actions to recover more easily.

Traditionally, almost all the enterprise systems are able to keep the historical data of the execution of business processes. This data can be used for building a predictive model that can be used to give recommendations.

1.1 Research goal

In order to apply prescriptive process monitoring on business processes in practice it is important to develop a toolset implementing prescriptive process monitoring approaches to support process participants in making decisions about what to do in order to prevent negative outcomes of ongoing process executions.

Therefore, the main goal of this research is focused on building a process-oriented recommender system providing recommendations about actions to carry on during the execution of a business process to maximize the probability of achieving a positive outcome in an ongoing process execution. In particular, based on this research goal we need to answer the research question “How can we build an interactive user interface for process-oriented recommender systems?”.

1.2 Relevant concepts

The University of Tartu has developed an open-source project to monitor business processes which is called Nirdizati. By using Nirdizati users can train predictive models and apply them in order to generate predictions about expected result of ongoing executions of a business process.

The same research group has developed a technique that uses these predictions to raise alarms for preventing unwanted results during the execution of a process, such as the complaint of a customer or a deadline missed. This specific method is called "alarm-based prescriptive process monitoring" because it is based on the idea of using alarms for prescribing when to apply a certain intervention to avoid a negative outcome in an ongoing process execution.

In addition to this technique, we implemented an approach for prescriptive process monitoring providing recommendations in terms of temporal rules to be satisfied in order to increase the probability of achieving a positive outcome like, for example, when an order is received eventually a discount should be applied to increase the probability of customer satisfaction.

2. Background

2.1 Process mining

Process mining is a research area which combines machine learning and data mining alongside analyzing and modeling processes [2]. This family of techniques are aimed at discovering, monitoring and improving business processes based on information taken from event logs that are available in enterprise systems.

Nowadays, enterprise systems hold huge amounts of data, however most of these data are stored in unstructured way. Extracting data is one of the main steps in process mining. Consider an enterprise system which handles requests. We can define each request as a different trace in the event log. In this case, each activity for handling a request can be defined as events and stored inside the trace. Since each activity matches with one event, the name of the activity is used for identifying this event. However, different events can have the same name. Therefore, we need other attributes to characterize events. In general, event logs keep for each event a timestamp. However, some additional attributes can be stored like resources, lifecycle transitions of activities or some other data elements.

Although there are several different process mining techniques, in general they are divided into three main branches including process discovery, conformance checking and enhancement [2]. Detailed information about each of them is given below.

1. **Process discovery** is a technique for creating a process model by using an event log therefore acquiring the behavior of the log [2]. There are two types of models that can be discovered:
    - Procedural: produces a procedural model so that each trace in the log should fit one path of the model.
    - Declarative: produces a set of constraints that should be satisfied by all the traces in the log.
2. **Conformance checking** is a technique for checking whether an event log matches with an existing model or not [2]. It takes an event log and a model as inputs and returns differences between them. By using this technique, it is possible to detect

potential deviations in the event log and calculate their severities. Two types of inconsistencies can be detected by using conformance checking:

- Unfitting log behavior: behavior detected in the log which is not accepted by the model.
- Additional model behavior: behavior allowed by the model never detected in the log.

3. **Enhancement** is a technique which is aimed at improving an existing model by using event logs [2]. While conformance checking is focused on comparing a log and an existing model, this technique is intended for improving an existing model. There are two possible ways for making improvements on a model.
    - Repair: It is about updating an existing model in order to increase its alignment with a log.
    - Extension: It is about extending an existing model by adding new information retrieved from logs.

2.2 Tools

The main purpose of process mining tools is analyzing execution logs. There are three main tools which can be used for analyzing logs: Disco, ProM and Apromore. The first and the second one are desktop applications, while the third one is a web application.

Disco – is designed for creating visual maps from process data [3]. The main features of this application are “Automated process discovery”, “Process map animation”, “Detailed Statistics about Logs”, “Log Filters”. Disco generates process maps from row data, then presents animations and statistics about these data. Moreover, its filters help users to easily extract process executions with different characteristics from the logs. The tool can import and export logs in different file formats such as XES, MXML, CSV and MS Excel.

Apromore – is an open-source web platform for process mining which has useful features such as “Performance mining”, “Process variants analysis”, “Conformance checking”, “Predictive process monitoring” and “Stage-based mining” [4]. Besides that, users can easily share their process mining results with other users by using its “shared workspace” feature.

ProM – is a cross-platform process mining framework which works as a combination of plug-ins [5]. All users of ProM can make contribution to the framework by developing their own plug-ins.

2.3 Data sources

For storing event logs, the very first approach had been MXML which was introduced in 2003 [2]. This format is based on XML. Although MXML was a successful approach, it was difficult to extend with new data types. Therefore, XES was introduced, which has a more flexible tag-based structure [6]. Currently, XES is the main standard format for storing logs which has been accepted by the "IEEE Task Force on Process Mining" in 2010 [7].

There are five primitive types in XES which are shown in Table 1.

Table 1: Matching between core and XES types

Core types	Corresponding XES types
Int	xs:long
Float	xs:double
Boolean	xs:boolean
String	xs:string
Date	xs:dateTime

An XES log is a set of traces where each trace consists of a sequence of events. In addition to that, all events are sorted based on their timestamps inside a trace. All events in a log contain three main attributes:

- Case ID: represented as a trace attribute "concept:name" is a unique value for identifying a trace.
- Event name: represented as an event attribute "concept:name" is used for defining the activity associated to an event.
- Timestamp: represented as an event attribute "time:timestamp" is an attribute for identifying the timestamp of an event.

The above three attributes are the minimum elements for creating event log, however in real log files there can be more attributes for both events and traces. An example of XES file is:

```
<?xml version="1.0" encoding="UTF-8" ?>
<log xes.version="1.0" xes.features="nested-attributes" openxes.version="1.0RC7">
    <trace>
        <string key="concept:name" value="1"/>
        <event>
            <string key="concept:name" value="Event 1"/>
            <date key="time:timestamp" value="2020-04-01T16:30:00+02:00"/>
        </event>
        <event>
            <string key="concept:name" value="Event 2"/>
            <date key="time:timestamp" value="2020-04-01T17:00:00+02:00"/>
        </event>
    </trace>
    <trace>
        <string key="concept:name" value="2"/>
        <event>
            <string key="concept:name" value="Event 1"/>
            <date key="time:timestamp" value="2020-04-02T16:30:00+02:00"/>
        </event>
        <event>
            <string key="concept:name" value="Event 3"/>
            <date key="time:timestamp" value="2020-04-02T16:45:00+02:00"/>
        </event>
        <event>
            <string key="concept:name" value="Event 2"/>
            <date key="time:timestamp" value="2020-04-02T17:00:00+02:00"/>
        </event>
    </trace>
</log>
```

Each attribute is described as a key-value pairs and are associated to an attribute type.

Another format for storing log data is CSV which stands for comma-separated values. This format is easy to manage with Python because it can be easily converted into pandas (library for Python) DataFrame format [8]. Also, it is easy to convert XES logs into CSV. While doing this, we have to consider that there are two types of attributes that we need to take into consideration:

- **Static attributes**: These are trace attributes which will be repeating values for each event inside the same trace.
- **Dynamic attributes:** These are event attributes and change for each event in a trace

Therefore, in CSV, we will have a table format dataset in which each row corresponds to an event except the first row which is a header and contains both static and dynamic attribute names. Static attributes are repeated with the same value in all the rows referring to events belonging to the same trace.

Below the same sample log file presented in CSV format where Trace ID is static, Concept name and Timestamp are dynamic attributes.

Trace ID, Concept name, Timestamp

1, Event 1, 2020-04-01T16:30:00+02:00

1, Event 2, 2020-04-01T17:00:00+02:00

2, Event 1, 2020-04-02T16:30:00+02:00

2, Event 3, 2020-04-02T16:45:00+02:00

2, Event 2, 2020-04-02T17:00:00+02:00

Example CSV format based on the sample log file

In the above examples (XES file and CSV), there are two traces and their IDs are 1 and 2 respectively. The first trace consists of two events (Event 1, Event 2) and the second trace consists of three events (Event 1, Event2, Event 3).

2.4 Declare

Declare is a declarative process modelling language that provides constraints which need to be satisfied over process executions [9].

Declare constraints are based on templates and applied to activities. The main groups of Declare templates are the followings [10]:

Existence – templates are applied on a single event for checking either the number of its occurrences in a trace or its position in the trace [11]. For all the existence templates, event A activates the constraint.

Table 2: Existence templates

Template	Method	Description	Activation
Existence	existence(n, A)	event A should occur at least n times in the trace	A
Absence	absence(n + 1, A)	event A should occur at most n times in the trace	A
Exactly	exactly(n, A)	event A should occur exactly n times in the trace	A
Init	init(A)	trace should start with event A	A

Choice – templates are applied on two events for checking if (at least) one of them occurs in a trace. For both the choice templates, either event A or event B can activate the constraint.

Table 3: Choice templates

Template	Method	Description	Activation
Choice	choice(A, B)	event A or event B should occur at least once in the trace	A, B
Exclusive choice	exclusiveChoice(A, B)	event A or event B should occur at least once in the trace but not both	A, B

Relation – templates are applied in order to check the relative position between two events. While event B activates the constraint for precedence, alternate precedence, and chain precedence (and A is the target), for the remaining templates event A activates the constraint (and B is the target) [11].

Table 4: Relation templates

Template	Method	Description	Activation
Responded existence	respondedExistence(A, B)	if event A occurs in the trace then event B should occur as well	A
Response	response(A, B)	if event A occurs in the trace then event B should occur eventually after it	A
Alternate response	alternateResponse(A, B)	if event A occurs in the trace then event B should occur after A and before A reoccurs	A
Chain response	chainResponse(A, B)	if event A occurs in the trace then event B should occur immediately after it	A
Precedence	precedence(A, B)	if event B occurs in the trace it should be preceded by A	B
Alternate precedence	alternatePrecedence(A, B)	if event B occurs in the trace it should be preceded by A and no B reoccurs in between	B
Chain precedence	chainPrecedence(A, B)	if event B occurs in the trace it should be preceded by A immediately	B

Negative Relation – templates are applied in order to check that two events do not occur in certain orders. While event B activates the constraint for not precedence and not chain precedence (and A is the target), for the remaining templates event A activates the constraint (and B is the target) [11].

Table 5: Negative relation templates

Template	Method	Description	Activation
Not responded existence	notRespondedExistence(A, B)	if event A occurs in the trace then event B should not occur	A
Not response	notResponse(A, B)	if event A occurs in the trace then event B should not occur after it	A
Not precedence	notPrecedence(A, B)	if event B occurs in the trace it should not be preceded by A	B
Not chain response	notChainResponse(A, B)	if event A occurs in the trace then event B should not occur immediately after it	A
Not chain precedence	notChainPrecedence(A, B)	if event B occurs in the trace it should not be preceded by A immediately	B

2.5 MP-Declare

MP-Declare is an extended version of Declare which provides the possibility to model constraints not only on the control flow but also on data and time. There are three types of conditions on data that can be defined [9]:

- **Activation conditions:** an activation condition must be satisfied when an activation of a constraint occurs. Example:
    - response(A,B) with A.number > 6 – means that when A occurs with number greater than 6, B should eventually occur after A.
- **Correlation conditions:** a correlation condition must be satisfied when a target of a constraint occurs and involves attributes of both activation and target. Examples:
    - response(A,B) with T.number < 9 – T refers to the target event and means that when A occurs, B should eventually follow with number lower than 9 (correlation condition only on the target event).
    - response(A,B) with T.number < A.number – is a correlation condition where A and T refers to activation and target event respectively. This constraint means that when A occurs, B should eventually follow and attribute number associated to B should be lower than attribute number associated to A (correlation condition on both activation and target events).
- **Temporal conditions:** specify time distances between activations and targets. Examples:
    - When event A occur, event T must subsequently occur within 1 minute.
    - When event A occur, event T must subsequently occur after 1hour.

3. Contribution

3.1 Prescriptive monitoring

Predictive process monitoring is a method for predicting the future state of a running process by using historical data which is stored in an event log. Existing methods can make predictions about the future state of a process. However, they cannot make prescriptions about when and how process participants need to interact with the system for decreasing the chance of undesired outcomes. In this research, we present two different approaches that we have implemented for prescriptive process monitoring, where together with predictions about the outcome of a process also recommendations are provided on what to do to decrease the probability of a negative outcome. The source code is publicly available in [12]. The first approach is called *alarm-based approach* and is designed for raising an alarm in case there is a higher probability for a bad outcome to occur. The second approach is a *Declare based approach* and relies on decision trees for providing recommendations in terms of Declare constraints that need to be satisfied/violated in order to maximize the probability of a positive outcome.

The input for both these approaches is an event log. Here we use XES which is the most common format for storing event logs. We do not use the whole event log as an input. Instead, we split it into two parts. The first part is called train dataset, while the another one is called validation dataset. In addition, in the alarm-based approach the training set is divided into training and validation set. We optimize the parameters of the algorithms for predictions using the validation set, then build our predictive model using the train dataset and then try to make recommendations on the validation dataset.

Although we implemented two different approaches, some of the initial steps are common for both of them.

The very first step is splitting the event log file in train and validation dataset. There are four splitting methods that user can select [13]:

- **Sequential order:** It is the simplest form of splitting. The event log is split by the given ratio without making any changes in the order of the traces.
- **Temporal order:** Firstly, traces are sorted based on the timestamp of their start event and after that splitting can be made on the event log by the given ratio.
- **Random order:** Firstly, traces should be sorted randomly and then the event log can be split by the given ratio.
- **Strict temporal order:** It is a complex form of temporal ordering. The first step is to apply a temporal ordering. Then it is needed to filter the train set in such a way that if the last event in a trace does not occur before the start event of the first trace in the validation set, the trace is filtered out.

For both of the implemented methods the input log is split. After the user selects a log split, depending on the user's choice on which approach he or she wants to use, these train and validation datasets are passed to the proper module of the back-end. In the following sub-sections, we describe what happens after this point for each of the approaches.

3.2 Declare-based approach

For the Declare-based approach, there are several steps that need to be followed in order to make recommendations. These steps are explained in this section.

3.2.1 Trace encoding

The very first step is encoding the traces in the train log. The encoding process starts with the application of an a-priori algorithm which is an association rule mining algorithm to identify frequent item sets in a dataset [14]. By using the a-priori algorithm, all pairs of distinct events appearing in the log are identified and for each pair, we determine how many times the two events occur together in the log traces. Then we compute the frequency of each pair by dividing number of traces in which the two events of the pair occur together by the total number of traces of log. Then, we filter these pairs for finding the most frequent ones. If the frequency of the pair is greater than a threshold which is defined by the user, it is considered to be frequent.

After that, for each frequent pair we instantiate the Declare templates: Responded Existence, Response, Alternate Response, Chain Response, Precedence, Alternate Precedence and Chain Precedence. These constraints represent the features that will be used to encode the traces in the train dataset before giving it as input to the algorithm for training the predictive model (a decision tree in this case). In particular, for each trace in the train dataset, we build a feature vector in which the elements represent the truth values of these constraints in the trace. For building the feature vectors, the constraints (the features) need to be checked with respect to the traces in the train datasets. To this aim, we implemented a checker

The result of the checker for each input trace and constraint contains several values.

- **number of fulfillments in the trace:** the number of constraint activations in the trace which are fulfilled.
- **number of activations in the trace:** the total number of constraint activations in the trace.
- **number of violations in the trace:** the number of constraint activations in the trace which are violated.
- **number of pending events in the trace:** the number of activations that are not fulfilled, but for which there exists a continuation of the trace that can fulfill them.
- **state of the trace:** This value depends on whether the input trace is complete or not. Besides that, the number of fulfillments, activations, violations and pending activations in the trace are taken into account. The trace can be violated, satisfied, possibly violated (in the case of an incomplete trace, the trace is currently violated but can be satisfied in the future) or possibly satisfied (in the case of an incomplete trace, the trace is currently satisfied but can be violated in the future). This output is made based on the conditions shown in Figure 1.

Template	Poss.viol	Poss.sat	Viol	Sat
response responded existence co-existence	$!done \wedge (p > 0)$	$!done \wedge (p = 0)$	$done \wedge (p > 0)$	$done \wedge (p = 0)$
not response not chain response precedence not precedence absence absence2 absence3 chain precedence not chain precedence alternate precedence	-	$!done \wedge (v = 0)$	$v > 0$	$done \wedge (v = 0)$
init	-	-	$v > 0$	$f > 0$
existence existence2 existence3	$!done \wedge (a < card)$	-	$done \wedge (a < card)$	$a >= card$
exactly1 exactly2	$!done \wedge (a < card)$	$!done \wedge (a = card)$	$(a > card) \vee (done \wedge (a < card))$	$done \wedge (a = card)$
not responded existence not succession not chain succession not co-existence	-	$!done \wedge (v = 0)$	$v > 0$	$done \wedge (v = 0)$
succession chain succession alternate succession chain response alternate response	$!done \wedge (v = 0) \wedge (p > 0)$	$!done \wedge (v = 0) \wedge (p = 0)$	$(v > 0) \vee (done \wedge (p > 0))$	$done \wedge (v = 0) \wedge (p = 0)$
choice	$!done \wedge (a = 0)$	-	$done \wedge (a = 0)$	$a > 0$
exclusive choice	$!done \wedge (a = 0)$	$!done \wedge (v = 0) \wedge (a > 0)$	$(v > 0) \vee (done \wedge (a = 0))$	$done \wedge (v = 0) \wedge (a > 0)$

Figure 1: How to decide the state of a trace for each Declare template *[15]*

done – is true if the trace is complete, otherwise false, p – number of pending activations in the trace, v – number of violations in the trace, f – number of fulfillments in the trace, a – number of activations in the trace

A trace is encoded by assigning to each feature in the feature vector the value of 0 if the corresponding constraint is violated or 1 if the corresponding constraint is satisfied. Note that since the traces in the train dataset are all complete, we can only have two possible values for the features (a constraint cannot be possibly satisfied or possibly violated in a complete trace).

3.2.2 Labeling

After trace encoding, we need to make labeling before passing the encoded traces as an input to the algorithm for training the decision tree.

In out experiments, we label traces as fast or slow. In order to decide whether a trace is fast or slow, firstly the difference between the timestamps of the last and the first event in the trace is calculated (the trace duration is computed). Then, if the difference is lower than a

threshold, as the trace is considered to be fast, otherwise it is slow. For getting the value of the time threshold, there are two possible options in our implementation. Either, it is defined by the user via an input field in the UI, or it is defined as the average time durations of the traces in the train log.

After the labeling is done, a new element is added to the feature vectors equal to 1 or 0 based on whether the trace is fast or slow, respectively.

3.2.3 Decision tree

Now, the predictive model can be trained. In particular, we train a decision tree [16]. We use a decision tree as predictive model, because decision trees are white box classifiers and can be used to explain the predictions. These explanations can then be used to extract the recommendations. To train the decision tree, the encoded traces and labels should be passed as an input to the decision tree construction algorithm. These are provided in the form of a matrix where the first row contains the feature names that are the Declare constraints used as features and the string “label”. The other rows are the encoded traces in the train log.

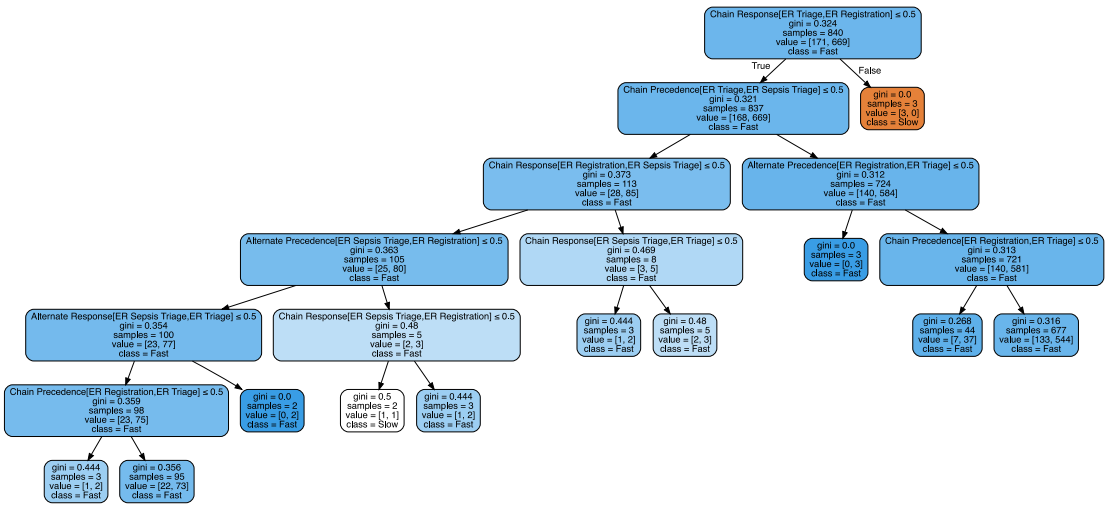

Figure 2: An example of a decision tree

Figure 2 shows an example of a decision tree. On each path there can be different nodes. Each node contains the feature name which is a Declare constraint, the gini value of that feature [16], the total number of samples corresponding to that path, number of slow and

fast samples corresponding to that path, and the predominant class value corresponding to that path, which can be either “Fast” or “Slow”.

3.2.4 Making recommendations

After generating the decision tree, all the positive paths are selected, i.e., the paths that end with a leaf node that has class value equal to “Fast”. Then, for each prefix of each trace in the validation set (i.e., an incomplete trace in the validation set) the tree can be queried to build recommendations.

For each positive path and partial trace, one recommendation can be built. In general, the total number of recommendations for one partial trace is equal to the total number of positive paths of the decision tree.

Each recommendation is made by taking into account the state of the constraints on the positive path taken into consideration and the state of these constraints in the partial trace. Table 6 shows how each recommendation is built based on the values of these states.

Table 6: Making recommendation due to the decision tree and a partial trace

The state of constraint on decision tree	**The state of constraint on partial trace**	**Decision**
Satisfied	Violated	There is a contradiction. This case is discarded.
Satisfied	Satisfied	Everything is good. No actions need to be taken.
Satisfied	Possibly violated	An action should be taken to satisfy this constraint.
Satisfied	Possibly satisfied	The upcoming actions must not violate this constraint.
Violated	Satisfied	There is a contradiction. This case is discarded.

Violated	Violated	Everything is good. No actions need to be taken.
Violated	Possibly violated	The upcoming actions must not satisfy this constraint.
Violated	Possibly satisfied	An action should be taken to violate this constraint.

Firstly, the constraints from the nodes on the positive path are taken. If the value for a constraint is less than or equal to 0.5 in a node, this means that this constraint should be violated. On the other hand, if the value for a constraint is greater than or equal to 0.5 in a node, this means that the constraint should be satisfied.

With this procedure we derive the values in the first column of Table 6. The values in the second column can be obtained by checking the constraint on the partial trace.

Two possible scenarios are possible while making a recommendation:

1. **Making recommendation is possible for a certain positive path in the tree:** While iterating over the constraints in the path, the states provided by the decision tree and the ones obtained by checking the constraints on the partial trace are never contradicting. The recommendation is the conjunction of the atomic actions recommended for each node in the path.
2. **Making recommendation is impossible for a certain positive path in the tree:** While iterating over the constraints in the path, the states provided by the decision tree and the ones obtained by checking the constraints on the partial trace are contradicting at least once. In this case, the positive path does not generate any recommendations.

The extracted recommendations can be ordered based on the confidence value of the prediction given in the corresponding positive path. The top ones are the most reliable ones because they express the conditions under which the positive outcome is most probable.

3.3 Alarm based approach

This approach firstly tries to predict whether there will be an undesired outcome or not and then in case there is a prediction about an undesired outcome it fires an alarm. This alarm is a recommendation for the user to make an intervention on the process execution to prevent the undesired outcome. In order to make such a prediction, a predictive model is built using the train dataset. Based on this model our system can make a decision whether to fire an alarm or not [17]. This alarm should be raised when the probability of the undesired outcome is higher than a threshold which is calculated via empirical way in this case. This approach is called "Basic Alarm System". The alarm needs to be linked with a system which presents suggestions for overcoming the issue such as calling a business analyst, an engineer and so on. In this example, the alarm is raised whenever the probability of the undesired outcome is higher than the threshold. However, this approach is not the best one in some cases such as when the probability of undesired outcome is not stable and changes after each event available in the partial trace. In this case, our basic alarm scenario fails because it stays raising even if the probability of undesired outcome decreases in the next event.

As a more complex form of alarm-based system, we can use a "Delayed firing system" in which the alarm is raised only if the probability of the undesired outcome remains higher than the threshold for k successive events [17].

We can improve even more the previous alarm system by changing it to the "Prefix-length-dependent Threshold System". The main advantage of this system is that, unlike the "Delayed firing system", it has multiple distinct thresholds based on the length of the prefix [18]. In particular, each threshold is optimized for one prefix length or a set of prefix lengths. This is useful because in the systems which have dynamic costs (for example an early intervention could have a lower cost than a late intervention), setting one threshold is not cost-effective. Instead, using multiple thresholds based on different prefix lengths is more profitable.

Lastly, the most complex form of alarm system is called “Multi-alarm system”. This system contains a set of alarms that have distinct compensation and intervention costs [18]. In case an undesired outcome is predicted, this type of system fires the alarm which has the lowest cost among others. Here, the task is to select the most appropriate alarm using multi-class classification. In particular, each alarm is a different class where no alarm is also considered as a separate class. Solving this classification task provides the best alarm to be raised. The best alarm-based approach is proven to be the one based on hierarchical thresholding [18]. Here, thresholds are trained in hierarchical order in order to make a decision for choosing an alarm. The strategy behind hierarchical thresholding is, again, to fire an alarm that has minimum compensation and intervention costs.

The starting point for any alarm-based prescriptive process monitoring approach is same as for the Declare-based approach. Firstly, the log is split in train and validation dataset.

However, unlike the Declare-based approach, in the alarm-based approach, the training is carried out by considering trace prefixes extracted by the traces of the train dataset. In addition, the train dataset is split in train and validation to allow hyperparameter optimization. Moreover, here all operations are made on pandas DataFrame. That is why the log files are converted into DataFrame via a custom-developed converter script. In this phase, some derived attributes are added in the form of separate columns into DataFrame for each event in the train and validation datasets:

- event_nr: Index of the event in the prefix
- month: Month in the timestamp of the event
- weekday: Weekday in the timestamp of the event
- hour: Hour in the timestamp of the event
- timesincemidnight: Time passed in minutes after midnight
- timesincelastevent: Time duration in minutes between previous and current event inside the trace. For the first trace this value is 0.
- timesincecasestart: Time duration in minutes between start and current event inside the trace. For the first trace this value is 0.

Then each prefix is labeled with a label:

- label: It describes whether the trace is fast or slow. The trace is fast if its execution time is less than the average trace execution time of the log.

After adding these attributes, in the DataFrame for each event, there is one row which contains all dynamic and static attributes that are related to this event. Within the same trace, each event has the same static attributes. The pandas DataFrame derived from the train dataset is used as input to train a predictive model. To this aim a part of the train dataset (20%) is used for hyperparameter optimization. The pandas DataFrame derived from the validation dataset is used for validating the alarm-based approach.

3.3.1 Predictive process monitoring part

The user can select either Random Forest (RF) or Light Gradient Boosting Machine (LGBM) as classifiers. Before training the classifiers hyperparameter optimization is performed. Hyperparameter optimization is intended for selecting the best hyperparameters to pass as input to the machine learning methods [19]. The train dataset is used for optimizing hyperparameters of the classifier by k-fold-cross-validation where k is equal to 3 in our case. While applying this, the whole event log is divided into k folds. Then all of the folds are used for training, except the last fold which is used for validation. This is repeated k times until each of these folds becomes a validation set. The final result is calculated on the result of k validations. In the implementation, the Hyperopt framework [20] is used for achieving hyperparameter optimization.

Optimized hyperparameters in the first step and train data are used for training the final classifier [19]. After that, predictions are generated.

3.3.2 Prescriptive process monitoring part

In the second step the user has several options to choose:

Basic mechanism: It is a basic way of seeking the alarm threshold:

1. optimizing threshold: The logic is based on empirical thresholding. Here the cost of undesired outcome and intervention are taken into account.
2. optimizing threshold effectiveness: Mitigation effectiveness is also considered in addition to the cost of undesired outcome and intervention.

3. optimizing threshold compensation: The cost of compensation is also considered alongside with the cost of undesired outcome and intervention.

Prefix-length mechanism: There are two possible options. The user can either select optimizing two prefix-length dependent thresholds or three prefix-length dependent thresholds. Using more than one threshold is intended to reduce cost.

Fire delay mechanism: Two options are provided:

1. optimizing fire delay: Here only one threshold is used. The logic is the combination of the basic mechanism for optimizing the threshold with the idea of fire delaying.
2. optimizing fire delay prefix length dependent threshold: Here multiple thresholds are used. The logic is the combination of optimizing the threshold with prefix length-dependent mechanism with the idea of fire delaying.

Hierarchical Thresholding: While the previous approaches are based on a single alarm, this approach is based on multiple alarms. In our project, it has been implemented via two alarms. So, firstly each of these two alarms are trained versus no alarm and then against each other via one-vs-one approach. In the end, the decision is made either the alarm with the lowest cost is fired or no alarm is fired.

4. Functional overview

In this section, we provide information about how our system works from the beginning to the end. There are three main subsections which are general, alarm-based approach, and Declare-based approach. Since each approach's workflow starts in the same way, in the general part, we present the steps which are common for both of them. After that, depending on the user's choice, the workflow continues in a different way depending on the selected approach. The entire workflow is shown in Figure 3.

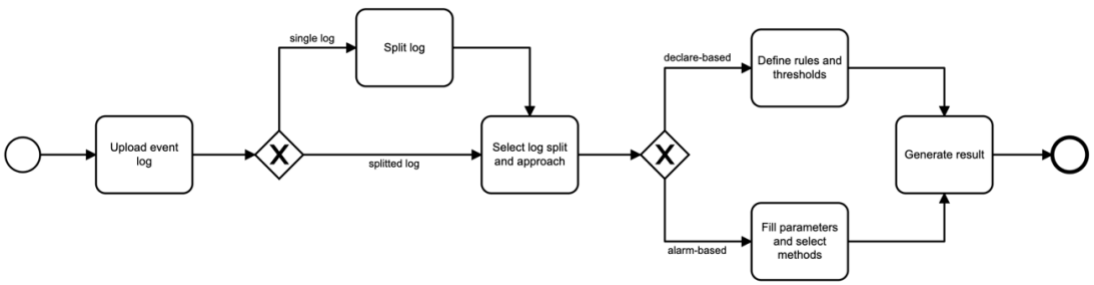

Figure 3: The workflow of the application

4.1 General

This general section contains four steps which the user needs to follow regardless of the approach he or she wants to select. These four steps are introduced below.

4.1.1 Upload Log

In order to interact with the system, the main input is an event log file in XES. Here the user can either upload a single log file and later split it or upload train and validation set separately. For uploading a single event log file, the user needs to click on choose file button and select a log file with the XES extension from his or her file system. After that the submit button is enabled and the user should click on it if he or she wants to upload this file into the system. The procedure is the same for uploading train and validation set separately.

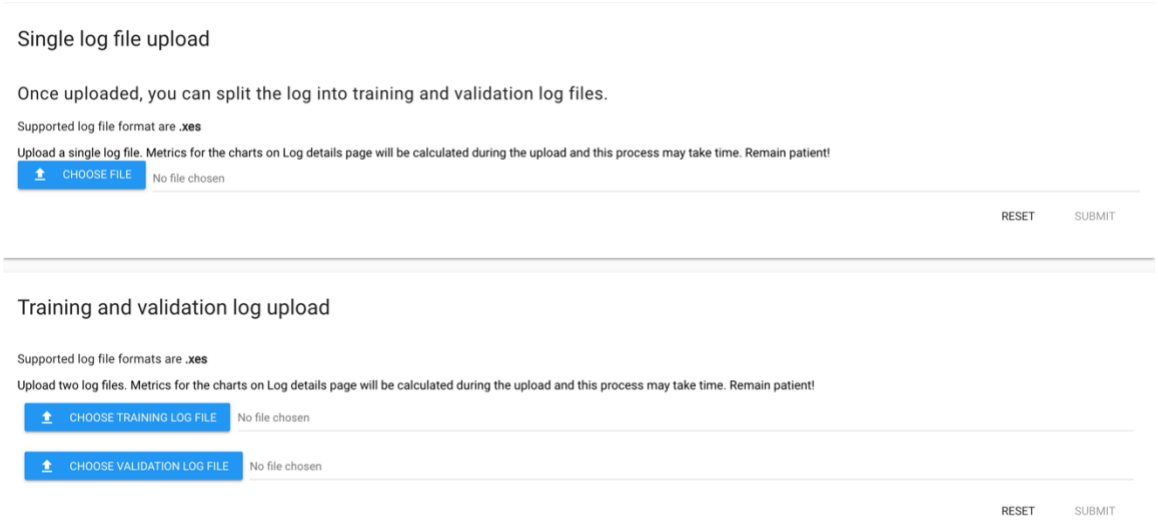

Figure 4: View of the upload component

4.1.2 Split Log

In case the user uploads a single event log file, the second step is splitting this log file into training and validation set. To achieve this, the user should navigate to the split view and find his or her previously uploaded log file in the select field on the left of the panel.

After that, the user should select a splitting method. There are four splitting options: “Sequential order”, “Temporal order”, “Random order”, and “Strict temporal order”.

Later, the user needs to choose a validation log percentage on the slider which is located on the right side of the panel. This value is used to define the splitting ratio of the event log. As a result, the log file will be split into training and validation set based on the given ratio. By default, this ratio is 0.2 which means that 80% of the log file will be used for creating the train set and the remaining 20% will be used for creating the validation set. In the end, after selecting these three values (log file, splitting method, validation log percentage), the user should press the submit button. Then the split request is sent to the back-end of the system and the splits are generated and stored so that they can be used in any other view of the system.

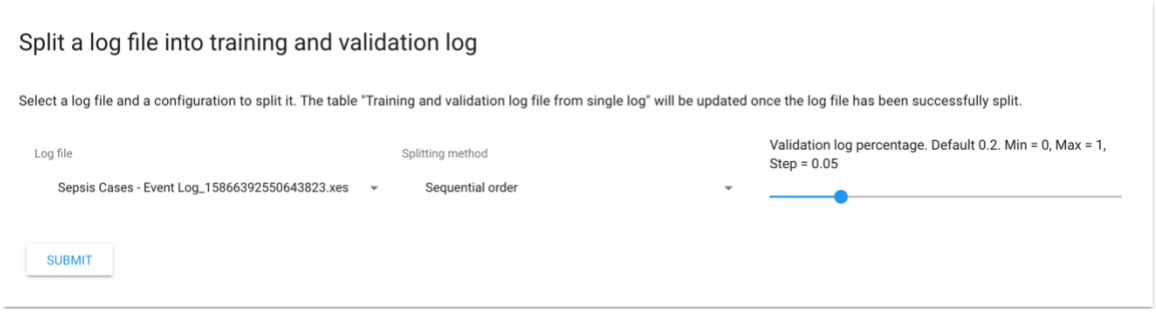

Figure 5: View of the split component

Now the user is ready to interact with the recommendation view. For this purpose, the user can choose the recommendation button on the left sidebar of the application. In this view, the top two components are in common for the two implemented approaches. While the first component is responsible for selecting the event log split, the second component is responsible for selecting the approach.

4.1.3 Select the log split

In this component, a list of log splits is presented in a select field. The user can find here the log splits that he or she have created in the previous steps.

Figure 6: View of the logs component

4.1.4 Select approach

After selecting the log split, the user needs to select one of the two implemented approaches. Depending on the user's choice the appropriate components will be shown below the approaches component.

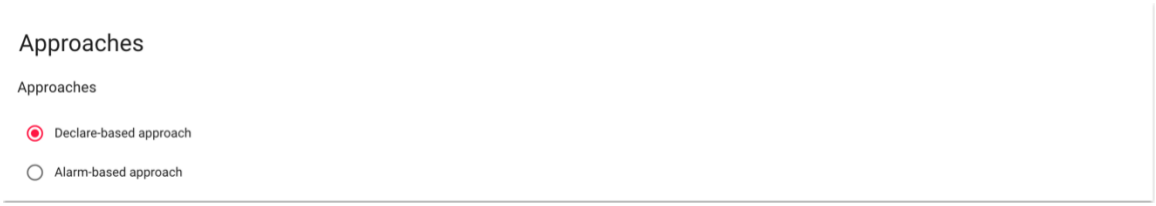

Figure 7: View of approaches component

4.2 Declare-based approach

In case the user selects the Declare-based approach, the components thresholds and rules, which are built for defining thresholds and rules, respectively, are presented.

4.2.1 Define thresholds

In this component, two types of thresholds need to be specified by the user. One is the frequency threshold, the other one is the time threshold.

The value for the frequency threshold is used for filtering pairs that are generated as a result of the a-priori algorithm. So, the pairs, which have a higher frequency than the user-defined frequency threshold, are chosen as frequent pairs.

Another threshold is the time threshold and its value is used when labeling the traces. If the total duration of the trace is higher than the user-defined time threshold the trace is slow, otherwise it is fast.

In case the user does not want to define a time threshold, he or she needs to activate the average time checkbox. In this case, the system will calculate the time threshold automatically based on the average trace duration.

Figure 8: View of the thresholds component

4.2.2 Define rules

While making trace encoding, it is possible to extend the Declare templates used to define features with data conditions. For this purpose, the user can specify these data conditions in the rules component. In particular, it is possible to specify an activation and a correlation condition to be applied to the predefined list of constraints used as features in the Declare-based approach.

These fields are not mandatory. So, the user can leave both of them or one of them empty. In such case, if the field is empty it means that there is no specific condition for the corresponding rule type. The rules should be defined in the decl format which was introduced in [21].

Figure 9: View of the rules component

After setting up all the parameters in the user interface, the user can click the generate button.

4.2.3 Display recommendations

After clicking the generate button, the request is sent to the back-end with the configuration values defined by the user. The validation set is used to generate recommendations over those traces (the traces in that set are considered as partial traces). After the generation of recommendations in the back-end the response is sent back to the front-end in JSON format. Since our system makes several recommendations for each trace due to different positive paths, in the result component different recommendations are presented for each of the traces.

The recommendations are presented as a list. Each recommendation has three values that are case ID, Impurity and decision.

Case ID is a unique value which identifies the trace. This is the value of "concept:name" in XES. Impurity is a value which describes the confidence of the recommendation. Decision presents the recommendation for the given Case ID.

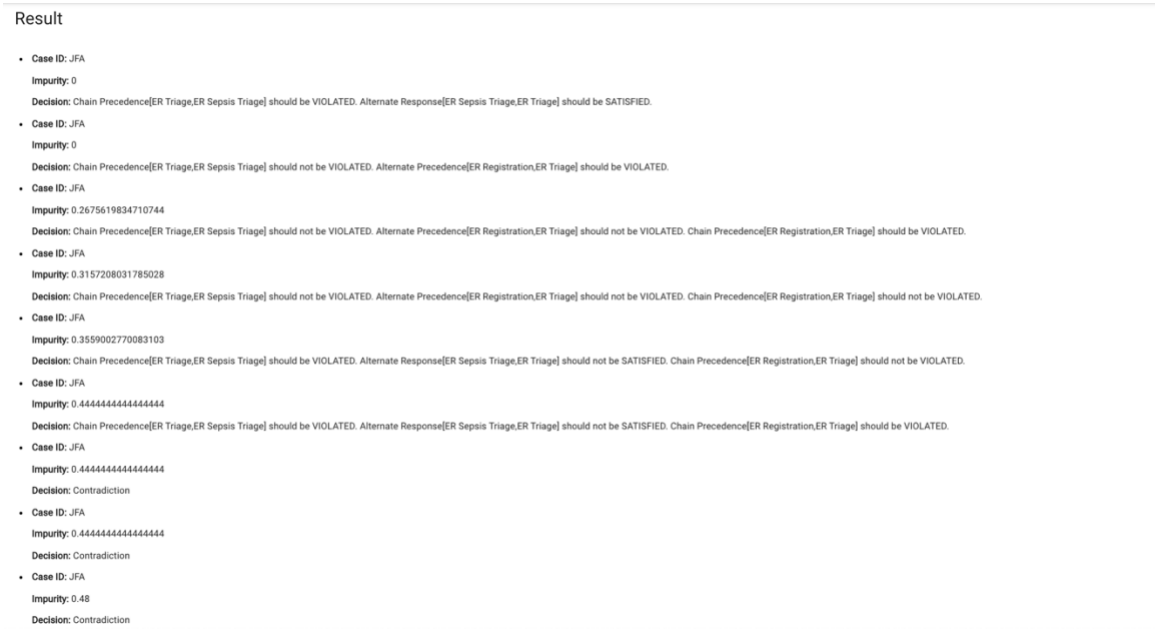

Figure 10: Result list in the Declare-based approach

4.3 Alarm-based approach

In case the user selects the alarm-based approach two components (methods and parameters) which are built for selecting methods and defining parameters respectively are presented.

4.3.1 Select methods

In this component, two select fields are presented. Each of them represents one step in the alarm-based approach.

The first field has two options that are RF and LGBM. When user selects one of them, the corresponding classifier will be used.

Unlike the first select field, the second select field has more options. Here the user can choose Basic mechanism, Prefix-length mechanism, Fire delay mechanism, or Hierarchical Thresholding. Depending on user's choice the corresponding threshold optimization and test scripts will be performed.

Figure 11: View of the methods component

4.3.2 Fill parameters

This component gets several parameters from the user via text fields that can be grouped as prefix length parameters and weights parameters.

Prefix length parameters contain three fields:

- min – is the value of the minimum prefix length (used for extracting prefixes from the train and the validation dataset)
- quantile – is used for calculating max prefix length. Firstly, the traces which have positive labels are selected and the number of events inside them is calculated and added into a list. Then the quantile of this list is calculated and rounded up for calculating the max prefix length.

- max – is the upper limit of the maximum prefix length. In case the calculated max prefix length is greater than max value then the max value is set to the max prefix length.

Weights parameters contain four fields that are the cost of undesired outcome, cost of intervention, cost of compensation and cost of postponing an intervention.

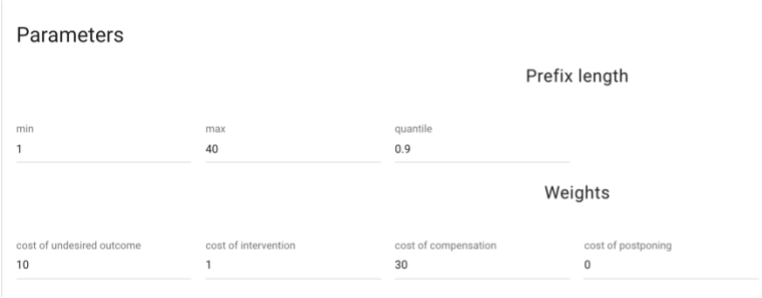

Figure 12: View of the parameters component

After selecting the methods and filling in all the text fields, the user can click the generate button.

4.3.3 Display result

After pressing the generate button, the request is sent to the back-end with the values defined by the user. After the generation of the result in the back-end, the response is sent back to the front-end in JSON format. The result is displayed as a table.

Dataset and method columns describe the name of the event log and the selected method, respectively. Columns c_miss, c_action, c_com, and c_ postpone provide the same values that the user set as weights in the parameter component which are the cost of undesired outcome, cost of intervention, cost of compensation and cost of postponing an intervention respectively. The scripts are performed in two different variants, which are described under the early_type column:

- **Const:** the value of the intervention cost does not change depending on the trace length.
- **Linear:** the value of the intervention cost increases depending on the trace length.

Some metrics and their values are also presented in the table under the metrics and value columns, respectively.

- Confusion matrix:
    - TP (True Positive) – number of correctly classified positive outcomes
    - FN (False Negative) – number of incorrectly classified positive outcomes
    - TN (True Negative) – number of correctly classified negative outcomes
    - FP (False Positive) – number of incorrectly classified negative outcomes
- Derived values from the confusion matrix:
    - Accuracy: ratio of correctly classified outcomes over the total number of outcomes
    - Precision: ratio of correctly classified positive outcomes over the total number of outcomes classified as positive
    - Recall: ratio of correctly classified positive outcomes over the total number of actual positive outcomes
    - F1 score: describes a balance between precision and recall and calculated with the formula: 2 * precision * recall / (precision + recall)
- Earliness: It states how early a prediction is given. A prediction given early is better because there is more time to act for preventing a negative outcome. Generally, it is calculated by this formula: (1 - (prefix number - 1) / case length).
    - Earliness mean: The mean of earliness values of different prefix lengths. This value is calculated on true alarms only.
    - Earliness std: Standard deviation of earliness values of different prefix lengths. This value is calculated on true alarms only.
    - Earliness alarms mean: The mean of earliness values of different prefix lengths. This value is calculated on all alarms.
    - Earliness alarm std: Standard deviation of earliness values of different prefix lengths. This value is calculated on all alarms.
- Cost: It describes total cost based on correct or incorrect predictions.
- Average cost: It is an average cost value over the number of prefixes.
- Baseline cost: It is the cost in case the system cannot predict the undesired outcome.

- Average baseline cost: It is an average baseline cost value that is calculated by dividing the baseline cost by the number of prefixes.
- Fire delay: The probability of the undesired outcome should be higher than the threshold for k-successive events to fire the alarm.

There are four possible cases providing different values for costs.

- There is no undesired outcome predicted and this prediction is correct. In this case, the cost is equal to 0.
- There is an undesired outcome, however it is not predicted. In this case, cost will be equal to the undesired outcome cost.
- There is an undesired outcome and it is predicted. In this case, the cost can be calculated by using the intervention cost and the undesired outcome cost. The calculation method may vary depending on the selected method.
- It is predicted that there will be an undesired outcome but in fact everything will be normal. Here the cost can be calculated by using intervention cost and compensation cost. The calculation method may vary depending on the selected method.

The last column is called thresholds and presents a tuple of thresholds. There is a prefix threshold value which is calculated by using Hyperopt framework [20] and based on this value, we can decide which threshold we can use depending on the prefix.

Result

dataset	method	metric	value	c_miss	c_action	c_postpone	c_com	early_type	threshold
log	fire_delay_prefix_length	fire_delay	1.0	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	prec	0.88	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	rec	0.1981981981981982	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	fscore	0.3235294117647058 4	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	tn	95	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	fp	3	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	fn	89	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	tp	22	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	earliness_mean	0.8555948312647336	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	earliness_std	0.0984468287821003	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	earliness_alarms_mean	0.863184321078183	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	earliness_alarms_std	0.0949160176822408 7	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	cost	25.287052124920944	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	cost_avg	0.1209906800235451 8	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	cost_baseline	27.073170731707318	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	cost_avg_baseline	0.1295367020655852 6	10	1	0	30	const	(0.669894804196026, 0.974108448596583)
log	fire_delay_prefix_length	fire_delay	7.0	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	prec	0.875	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	rec	0.06306306306306306	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	fscore	0.1176470588235294	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	tn	97	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	fp	1	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	fn	104	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	tp	7	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	earliness_mean	0.581953734671126	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	earliness_std	0.0854924899040084 4	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	earliness_alarms_mean	0.5961660395763657	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	earliness_alarms_std	0.0887735587888114 2	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	cost	26.992057875846317	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	cost_avg	0.1291486022767766 3	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	cost_baseline	27.073170731707318	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)
log	fire_delay_prefix_length	cost_avg_baseline	0.1295367020655852 6	10	1	0	30	linear	(0.21836088813080777, 0.6755940729378977)

Figure 13: Result table of fire delay prefix length dependent threshold

5. Tool implementation

We have implemented our recommender system as a web application. In this section we present the detailed information about the implementation of the application.

5.1 Architecture

While the back-end side is responsible for making operations on the event logs, the front-end side is targeted for establishing interactions with users.

The core project for our research is Nirdizati, a web application for predictive process monitoring. Our target is to add a new module which is called recommendation into it. We decided to create a separate back-end for this module. The front-end, instead, was integrated inside the Nirdizati UI. The architecture is shown in Figure 14.

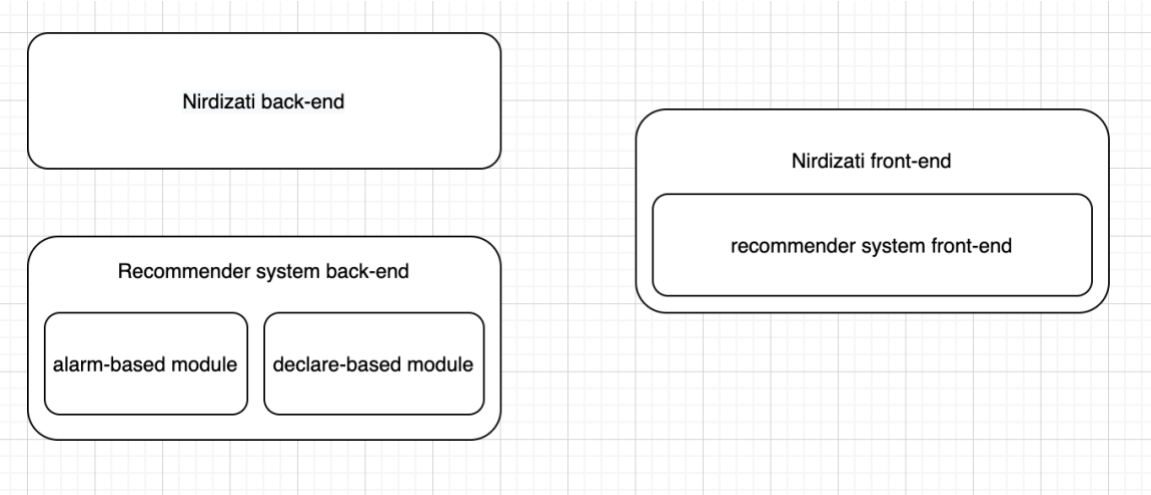

Figure 14: Architecture of the whole project

5.1.1 Back-end

The back-end side is implemented by using the REST framework and the architecture we have used in the back-end side is module based. We have implemented one module for each approach. While the first module implements the Declare-based approach, the second one implements the alarm-based approach. They take the user-defined parameters from the front-end side, download the log split from the Nirdizati database and then start processing the log split based on the appropriate scripts. In the end, it returns a JSON response to the front-end side.

5.1.2 Front-end

Regarding the front-end side, we divided the parts of the UI into components and make a component-based architecture for its implementation. There are two common components for both of the modules which are logs and approaches components respectively that are developed for selecting log splits and the approach. Also, there are two different result components one for each module which display the output depending on the approach. The other components are thresholds and rules for the Declare-based approach, methods and parameters for the alarm-based approach. There is a generate button the user needs to click after filling all the required inputs so that all the user-defined values are transferred to the back-end for processing. In the end, a response is returned to the front-end and the result component is filled for displaying the output to the user.

5.2 Technologies

After deciding to implement a web application, the next step was to decide the technologies to use.

5.2.1 Back-end

Based on our research, we realized that Python [22] was the best choice for us, because it has very rich machine learning and process mining libraries. One of the main libraries we use in this project is PM4Py [23] which is a very efficient tool for doing operations on event log files. Since it is strongly recommended using Python v3.6 for this library in its documentation, we built our project using this version of Python.

In addition, we used a well-known machine learning library which is called scikit-learn [24]. In particular, we have used scikit-learn v0.22.1 that was the latest one when we started the implementation of our project.

To achieve a better architecture, we decided to use Django [25] for building the back-end and the Django REST framework [26] for developing its API side. In particular, the back-end side of the application is built on Django v3.0.3 and Django Rest Framework v3.11.0.

5.2.2 Front-end

Since, the Nirdizati front-end was implemented in React [27], we also used the same framework and architecture for the front-end. For managing the state of the application, we used Redux [28], and AJAX for communicating with the back-end. For designing the user interface react-md [29] was used.

6. Evaluation

6.1 Data sets

In this section, we evaluate our project by using two different real-life event logs: Sepsis Cases and Prepaid Travel Cost. The data of Sepsis Cases were collected through the ERP system of a hospital and stored into an event log file that contains 1050 traces and 15214 events [30]. The second log file has 2099 traces and 18246 events that are related to travelling costs for two years which is called prepaid travel cost [31].

We split both of the log files into train and validation set with 0.2 ratio using sequential order (80% of the log for the train set, 20% for the validation set).

6.2 Declare-based approach

The output of this approach is a list of recommendations that contain three values. The first one is Case ID which is a unique value for identifying the trace. The second value is impurity that can be a value between 0 and 1 that describes the confidence of the recommendation. The last one is the text which shows the recommendation on a partial trace.

While making the evaluation on Sepsis Cases, we set the frequency threshold of pairs to 0.99, whereas on Prepaid Travel Cost, we set this value to 0.95. Regarding to the rules, we keep them empty for both of them that means constraints on events did not use any data conditions.

The result on the Sepsis Cases event log is presented below.

The number of positive paths from the decision tree is 9 and, which indicates that for each trace in the validation set we will have 9 recommendations (including contradictions). So, in total there will be 210 * 9 = 1880 recommendations. Due to the high number of recommendations, we are not able to put all of them inside this document. Instead, we only show the ones related to trace with case ID “JFA”.

Case ID: JFA, **Impurity:** 0

Decision: Chain Precedence[ER Triage,ER Sepsis Triage] should be VIOLATED. Alternate Response[ER Sepsis Triage,ER Triage] should be SATISFIED.

Case ID: JFA, **Impurity:** 0

Decision: Chain Precedence[ER Triage,ER Sepsis Triage] should not be VIOLATED. Alternate Precedence[ER Registration,ER Triage] should be VIOLATED.

Case ID: JFA, **Impurity:** 0.2675619834710744

Decision: Chain Precedence[ER Triage,ER Sepsis Triage] should not be VIOLATED. Alternate Precedence[ER Registration,ER Triage] should not be VIOLATED. Chain Precedence[ER Registration,ER Triage] should be VIOLATED.

Case ID: JFA, **Impurity:** 0.3157208031785028

Decision: Chain Precedence[ER Triage,ER Sepsis Triage] should not be VIOLATED. Alternate Precedence[ER Registration,ER Triage] should not be VIOLATED. Chain Precedence[ER Registration,ER Triage] should not be VIOLATED.

Case ID: JFA, **Impurity:** 0.3559002770083103

Decision: Chain Precedence[ER Triage,ER Sepsis Triage] should be VIOLATED. Alternate Response[ER Sepsis Triage,ER Triage] should not be SATISFIED. Chain Precedence[ER Registration,ER Triage] should not be VIOLATED.

Case ID: JFA, **Impurity:** 0.4444444444444444

Decision: Chain Precedence[ER Triage,ER Sepsis Triage] should be VIOLATED. Alternate Response[ER Sepsis Triage,ER Triage] should not be SATISFIED. Chain Precedence[ER Registration,ER Triage] should be VIOLATED.

Case ID: JFA, **Impurity:** 0.4444444444444444, **Decision:** Contradiction

Case ID: JFA, **Impurity:** 0.4444444444444444, **Decision:** Contradiction

Case ID: JFA, **Impurity:** 0.48, **Decision:** Contradiction

The result on the Prepaid Travel Cost event log is presented below.

The number of positive paths from the decision tree is 5 and it indicates that for each trace in the validation set we will have 5 recommendations (including contradictions). So, in total there will be 420 * 5 = 2100 recommendations. We show here recommendations related to the trace with case ID “request for payment 63682”.

Case ID: request for payment 63682, **Impurity:** 0

Decision: Alternate Response[Request For Payment SUBMITTED by EMPLOYEE,Payment Handled] should be VIOLATED. Chain Precedence[Request Payment,Payment Handled] should be VIOLATED.

Case ID: request for payment 63682, **Impurity:** 0.2603550295857988

Decision: Alternate Response[Request For Payment SUBMITTED by EMPLOYEE,Payment Handled] should not be VIOLATED. Chain Precedence[Request Payment,Payment Handled] should be VIOLATED.

Case ID: request for payment 63682, **Impurity:** 0.44197045389632195

Decision: Alternate Response[Request For Payment SUBMITTED by EMPLOYEE,Payment Handled] should not be VIOLATED. Chain Precedence[Request Payment,Payment Handled] should not be VIOLATED.

Case ID: request for payment 63682, **Impurity:** 0.45674740484429066

Decision: Contradiction

Case ID: request for payment 63682, **Impurity:** 0.49777777777777776

Decision: Contradiction

6.3 Alarm-based approach

For this approach, in the hyperparameter optimization part, the user can choose either RF or LGBM. In the threshold optimization part, there are 8 possible options including basic threshold optimization, prefix length dependent thresholds, basic fire delay, fire delay prefix length dependent threshold, and hierarchical thresholding. So, in total, there are 16 combinations. Besides that, the user can change the parameters related to prefix length and weights.

Since the number of combinations is high, we are not able to present each of them here. Instead, for the evaluation we choose LGBM for hyperparameter optimization and hierarchical thresholding for threshold optimization. While defining weights, we set the cost of undesired outcome, cost of intervention, cost of compensation and cost of postponing an intervention to 10, 1, 30, 0 respectively.

Table 7: Evaluation result of alarm-based approach

	Sepsis Cases	**Prepaid Travel Cost**
TN	83	107
FP	16	28
FN	75	26
TP	36	259
Precision	0.6923076923076923	0.9024390243902439
Recall	0.32432432432432434	0.9087719298245615
F-score	0.44171779141104295	0.9055944055944056
Earliness mean	0.9782776435856148	0.9843161343161343
Earliness std	0.06465720712076126	0.08826442061348375
Earliness alarms mean	0.9740275633879314	0.971045447003635
Earliness alarms std	0.07553162835270405	0.1246631202923554
Cost	29.878048780487806	25.24878048780488
Average cost	0.14227642276422764	0.06011614401858305
Baseline cost	27.073170731707318	69.51219512195124
Average baseline cost	0.1289198606271777	0.16550522648083626

The explanation of values inside the above Table 7:

- **TN, FP, FN, TP:** These values are the results of the confusion matrix
- **Precision**, **Recall**, **F-score:** These values are calculated based on the confusion matrix and describe the performance of the classification tasks
- **Earliness:** It describes how early an undesired outcome is predicted. While earliness mean and std is calculated based on cases where the alarm is fired correctly, earliness alarms mean and std are calculated based on all cases in which an alarm is fired.
- **Cost:** While cost and average cost are calculated based on whether the undesired outcome is correctly predicted or not, the baseline cost is calculated in case the undesired outcome cannot be predicted. The average values are calculated by dividing the costs by the number of traces.

7. Related work

Several techniques exist for implementing recommender systems [32]:

- **Content-based:** While the user interacts with a system, some data are collected which are related to his or her actions. Based on these data, the system can learn what makes this user satisfied or dissatisfied. So, based on this content, the system recommends items to the user that he or she will most probably like.
- **Collaborative filtering:** This technique is the most popular type of recommender system. The user gets item recommendations based on the preferences of other users who have the same interests. The interest relation between two users is calculated based on the history of their interactions with the system.
- **Demographic:** The items are suggested to the user based on his or her demographic profile. Some properties such as age, language, country, etc. are taken into account from the user's profile.
- **Knowledge-based:** The system recommends items to the user based on a particular domain knowledge. For instance, the system takes the problem description from the user and matches it with the recommendations related to solutions to the same problem.
- **Community-based:** The recommendation of items is given to the user depending on his or her friends' interests.
- **Hybrid systems:** It can be a combination of any of the above techniques. Hybrid systems have several hybridization ways [33]:
    - **Weighted:** Each used recommendation technique has different weights and the rank of recommendations is calculated based on these weights.
    - **Switching**: Depending on the item, the system switch between recommendation techniques.
    - **Mixed**: Recommendations that are made based on different recommendation techniques are presented together at the same time.
    - **Feature combination:** Features are combined together from different sources and passed to the recommender for making recommendations based on them.

- **Cascade:** The second recommendation technique refines the recommendations from the first one.
- **Feature augmentation:** The first technique makes the classification of items and this classification result is passed as an input to the second technique. It increases the quality of recommendations significantly.
- **Meta-level:** The first recommender generates a model and this generated model is used as an input for the second one.

In addition to this, in [33], common problems that are related to the recommender systems are presented including the new user and the new item problems.

The problem regarding to a new user is related to the fact that the system cannot recommend anything to him or her because there is not enough data available for a new user On the other hand, when the item is new, it is also difficult to recommend it to users because there is not enough data about users' interest on it. In [34], the author has introduced the PAR system and defined this system as a software which is intended to observe process executions, make predictions about their outcomes, and generate recommendations for making intervention to decrease failure risk. The project is built on a knowledge-based recommendation technique. The validation of the system is done by using data from UWV. UWV is an agency which supports people who are unemployed and are looking for a new job.

8. Conclusion

8.1 Summary

In this research, we have implemented a tool for providing alarm-based and Declare-based recommendations. To make our implementation accessible for different kinds of users, we developed an interactive web user interface for establishing an easy way of interaction between the users and the recommender system. We specifically focused on a web application integrated in the Nirdizati research project which provides different features for prescriptive process monitoring.

8.2 Future Work

Since this is the first version of the system, several improvements can be done as future work.

The very first thing that can be done as an improvement is integrating the back-end of our recommender-system with Nirdizati by adding a recommendation module inside the Nirdizati back-end itself. Since the UI is already joined with the Nirdizati front-end, it is reasonable to join the back-end sides as well.

The second improvement is related to the output of the alarm-based approach. Currently, it displays a table which contains several values inside. Other types of representations for the results could be considered for improving their understandability.

The third improvement is related to the output the Declare-based recommender system. In particular, the atomic actions suggested in a single recommendation could be aggregated by suggesting a sequence of activities to execute to maximize the probability of a positive outcome.

Finally, a user evaluation should be conducted to test the usability of our system.

9. References

[1] M. M. Fabrizio, D. F. Chiara, D. Marlon and G. Chiara, Predictive Monitoring of Business Processes, 2014.

[2] W. M. v. d. Aalst, Discovery, Conformance and Enhancement of Business Processes, 2011.

[3] "Disco," [Online]. Available: https://fluxicon.com/disco.

[4] "Apromore," [Online]. Available: https://apromore.org.

[5] "ProM," [Online]. Available: http://www.promtools.org.

[6] "XES standard," [Online]. Available: http://www.xes-standard.org.

[7] "Process mining," [Online]. Available: http://www.processmining.org.

[8] "Pandas," [Online]. Available: https://pandas.pydata.org.

[9] K. Denizalp, Rule Mining with RuM, 2019.

[10] B. Andrea, M. M. Fabrizio and S. Alessandro, Conformance checking based on multi-perspective declarative process models, 2015.

[11] S. Christian, F. Myriel and S. Stefan, Full Support for Efficiently Mining Multi-Perspective Declarative Constraints from Process Logs, 2019.

[12] S. Aladdin and M. M. Fabrizio, "Recommender system," [Online]. Available: https://bitbucket.org/shikhizada/prescriptive-monitoring.

[13] T. Kasekamp, A Web Application to Support Researchers in Predictive Process Monitoring Tasks, 2018.

[14] H. Markus, The Apriori Algorithm – a Tutorial, 2008.

[15] M. M. Fabrizio, M. Marco and B. Ubaier, Compliance Monitoring of Multi-Perspective Declarative Process Models, 2019.

[16] R. Lior and M. Oded, Decision Trees, 2005.

[17] S. A. Fahrenkrog-Petersen, N. Tax, I. Teinemaa, M. Dumas, M. d. Leoni, F. M. Maggi and M. Weidlich, Fire Now, Fire Later: Alarm-Based Systems for Prescriptive Process Monitoring, 2019.

[18] S. A. Fahrenkrog-Petersen, Decision Making in Prescriptive Process Monitoring, 2018.

[19] I. Teinemaa, Predictive and Prescriptive Monitoring of Business Process Outcomes, 2019.

[20] "Hyperopt," [Online]. Available: http://hyperopt.github.io/hyperopt/.

[21] V. Skydanienko, Data-aware Synthetic Log Generation for Declarative Process Models, 2018.

[22] "Python," [Online]. Available: https://www.python.org.

[23] "PM4PY," [Online]. Available: https://pm4py.fit.fraunhofer.de .

[24] "scikit-learn," [Online]. Available: http://scikit-learn.org.

[25] "Django project," [Online]. Available: https://www.djangoproject.com.

[26] "Django rest framework," [Online]. Available: https://www.django-rest-framework.org.

[27] "React," [Online]. Available: https://reactjs.org.

[28] "Redux," [Online]. Available: https://redux.js.org.

[29] "React-md," [Online]. Available: https://react-md.mlaursen.com.

[30] "Sepsis Cases," [Online]. Available: https://data.4tu.nl/repository/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460.

[31] "Prepaid Travel Cost," [Online]. Available: https://data.4tu.nl/repository/uuid:5d2fe5e1-f91f-4a3b-ad9b-9e4126870165.

[32] R. Francesco, R. Lior and S. Bracha, Recommender Systems Handbook, 2010.

[33] R. Burke, Hybrid Recommender Systems: Survey and Experiments, 2002.

[34] D. Marcus, d. L. Massimiliano, W. V. d. Aalst and H. A. Reijers, What if Process Predictions are not followed by Good Recommendations?, 2019.

License

Non-exclusive license to reproduce thesis and make thesis public

I, Aladdin Shikhizada,

1. herewith grant the University of Tartu a free permit (non-exclusive license) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,
   A Tool for Prescriptive Monitoring of Business Processes,
   supervised by Fabrizio Maria Maggi.
2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons license CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive license does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Aladdin Shikhizada

15/05/2020