Designing and Implementing a Framework for Event-based Predictive Modelling of Business Processes

Jörg Becker, Dominic Breuker, Patrick Delfmann, Martin Matzner

Department of Information Systems
University of Muenster - ERICS
Leonardo-Campus 3
48149 Münster
{becker,breuker,Delfmann,matzner}@ercis.uni-muenster.de

Abstract: Applying predictive modelling techniques to event data collected during business process execution is receiving increasing attention in the literature. In this paper, we present a framework supporting real-time prediction for business processes. After fitting a probabilistic model to historical event data, the framework can predict how running process instances will behave in the near future, based on the behaviour seen so far. The probabilistic modelling approach is carefully designed to deliver comprehensible results that can be visualized. Thus, domain experts can judge the predictive models by comparing the visualizations to their experience. Model analysis techniques can be applied if visualizations are too complex to be understood entirely. We evaluate the framework's predictive modelling component on real-world data and demonstrate how the visualization and analysis techniques can be applied.

1 Motivation

Enterprises increasingly invest in data analysis capabilities to gain competitive advantage [PF13]. Areas of application are manifold and numerous fields surrounding this topic emerged [CS12]. Not surprisingly, business process management (BPM) scholars have recognized the merits of data analysis long ago. Most importantly, process mining [Aa11] has emerged as a discipline developing methods for the analysis of business process event data collected from an organization's information systems. Typical process mining starts with discovering a process model and proceed with further analyses, for instance to eliminate problems with the process's implementation [ARW07].

Process mining is a discipline concerned mainly with techniques for retrospective data analysis [Aa11]. Hence, its techniques are applied with the same goals traditional business intelligence (BI) techniques are used—enabling knowledge workers to make good decisions fast [CDN11]—but the techniques are tailored to the BPM domain. However, BI is also increasingly used for operational support, i.e. in real-time settings [CDN11]. In BPM, complex event processing (CEP) software can be used to monitor events in real-time and trigger suitable actions when predefined patters are detected [EN10].

With analysis of the past and the present being important BPM topics, it is not surprising that analysis of the future is gaining momentum. Process mining scholars emphasize the opportunities of using event data to predict future events of running process instances, for instance to recommend appropriate actions [APS10]. Architectures to integrate event-driven process analytics with technologies such as CEP have been proposed [SMJ13]. Approaches to predict the completion time of process instances based on their current states have been developed recently [ASS11].

Motivated by the attention paid to predictive modelling in BPM, we present a framework for event-based predictive modelling of business processes. The framework requires the existence of historical event data and uses it to fit a probabilistic model. By feeding real-time data of running process instances into the framework, it allows evaluating probabilistically how the process will behave in the future. To implement this functionality, we draw on machine learning research [Mu12] and, in particular, grammatical inference, which is a discipline concerned with learning formal languages from data [Hi10]. The ultimate goal is to provide a technical basis for implementing process intelligence approaches capable of anticipating opportunities and threads before they occur, which gives managers a window of opportunity to take suitable actions. For instance, such functionality is a core part of modern intelligent BPM suites [JSC14]. These tools however require processes to be modelled explicitly, whereas our approach follows a process mining approach in which an appropriate model is learned from the data.

In predictive modelling projects, it is often necessary to convince domain experts with non-technical background that a predictive model will do more good than bad [PF13]. Hence, predictive modelling techniques should ideally produce comprehensible, interpretable models. Many techniques though do not lend themselves to interpretation. Thus, comprehensible models are sometimes used even if their predictive performance is worse than that of a black-box model [SK11]. We carefully design our framework such that the probabilistic models can be transformed into conceptual models in notations used frequently in BPM. This ensures that predictive models can be visualized in a form domain experts are familiar with. As a consequence, they can view and analyse these models to compare them with their domain knowledge.

The framework provides means to visualize predictive models at varying levels of detail. While this abstraction mechanism can provide a small, easy to read visualization at a high level of abstraction, a user may want analyse a visualization in more detail. These models can easily get too complicated to be understood. Fortunately, BPM scholars have long been concerned with managing and analysing large quantities of complex process models [Aa13]. For this reason, we integrate model analysis functionality into the framework. It allows domain experts to work even with complex models.

The remainder of this paper is structured as follows. Section 2 describes and justifies the design decisions we made when conceptualizing our framework. In section 3, we present results of an evaluation with real-world event data and illustrate the use of the visualization techniques with an example. Related work is discussed in section 4, while section 5 concludes and provides an outlook to future research.

2 Event-based Prediction Framework

2.1 Overview of the Framework

Before introducing the components of the framework in detail, we present an overview of its architecture (cf. figure 1). There are two data sources. The first is historical event data from finished process instances. The other source is real-time data from running process instances. Once a running process instance is completed, its event sequence should be added to the historical data. We use a simple data format that corresponds to event logs as they are used in process mining [Aa11]. Conceptually, an event log $X = \{x^{(1)}, \dots, x^{(C)}\}$ consists of C sequences of events. Each event sequence $x^{(c)}$ is a tuple $(x_1^{(c)}, \dots, x_{t_c}^{(c)})$ of t_c events. Thus, t_c is the length of event sequence $x^{(c)}$, which may be different for each sequence. Each event uniquely belongs to an event type $x_t \in \{1, \dots, E\}$. There are E different event types.

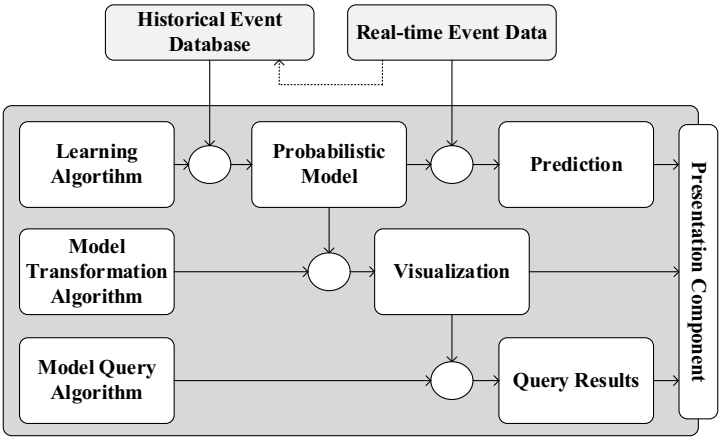

Figure 1: Overview of the Event-based Prediction Framework

Using the framework starts with selecting an appropriate subset of the event data. This is a creative, application-specific task and its support is not within the scope of our framework. Once a dataset is created, a learning algorithm is used to fit a probabilistic model that (hopefully) represents the event data well. The result delivered by the algorithm is a representation that can be used to make predictions. To this end, event data from running process instances must be fed into the framework. Given a sequence of events observed so far, the probabilistic model allows evaluating the probabilities of the process continuing with certain other event sequences. In particular, this functionality can be used to predict the most likely type of event with which a process will continue.

A user of the framework may want to make sure the predictive model does not contradict common sense. This analysis starts with deriving a conceptual model from the probabilistic model using a transformation algorithm. This conceptual model can be visualized and interpreted. It allows a user to compare the model with his expectations. As a result, he may either conclude that the probabilistic model is adequate or he may identify be-

haviour that contradicts his expectations. In the latter case, problem analysis could follow to find out if the probabilistic model is inadequate or if the expectations were wrong.

When the conceptual model generated by the model transformation algorithm is too complex to be read easily by humans, the framework provides algorithmic support to its users. In particular, model query algorithms can be used to determine whether certain patterns are found in the model. The patterns can represent model structure that a user either expects to find or expects not to find. By comparing this expectation with the query results, a model can be evaluated even if it is highly complex.

All components implemented are summarized in table 1 and described in detail in the following subsection. The framework is implemented entire in Java and is publicly available at https://github.com/DominicBreuker/RegPFA. It is meant to be used as a library and provides only rudimentary support for visualization. This paper describes the current state of the implementation and the rationale behind the framework's design, which should be understood as work in progress rather than a final version.

Table 1: Summary of the current components

Area	Component	Description
Learning algorithms	*EmMapLerner*	Learns a regularized PFA with EM-based MAP parameter estimation combined with grid search.
Model transformation algorithms	*Threshold pruning*	Visualizes a PFA at varying levels of detail.
	Automaton minimization	Simplifies the structure of an automaton obtained with *threshold pruning*.
	Petri net synthesis	Creates a petri net from an automaton and ensures that both models describe the same behavior.
Model query algorithms	*Subgraph pattern search*	Decides whether a subgraph pattern is part of a model created by a model transformation algorithm.

2.2 Learning Algorithms and Probabilistic Models

Given event data in the form described in section 2.1, modelling the event data probabilistically means finding a representation that specifies a probability distribution over the event sequences in the event log. Hence, the first design decision in our framework is to define what kind of probabilistic model should be used. Approximating distributions over sequences of symbols (~events) from a discrete alphabet (~set of event types) is a problem extensively researched in the field of grammatical inference [Hi10]. For this reason, grammatical inference constitutes the starting point of our discussion.

While numerous different probabilistic models are considered in the literature, the two best known are the Hidden Markov Model (HMM) and the Probabilistic Finite Automaton (PFA) [VEH13]. The commonality of both models is that the process generating the events is explained with a process of traversing an invisible state space. At each discrete step in time, the process is in exactly one state, and the event observed depends on that state. In the HMM [Ra89], the state space traversal is independent of the events, i.e., given a current HMM state, the next state depends only on the current state. A PFA [VTH05] is different. The next state depends on the current state and the current event.

For business processes, the PFA is more adequate than the HMM. To illustrate why, consider the simple example business process in figure 2 (a), represented as a business process model and notation (BPMN) diagram. Assuming that the performance of each BPMN activity corresponds to observing a corresponding event, the possible event sequences are *ABE*, *ACDE*, and *ADCE*. Their probabilities are 0.3, 0.35 and 0.35. A corresponding PFA is illustrated in figure 2 (b). By comparing the two models, it is obvious how the PFA's states can be interpreted as states of the BPMN process. For instance, the PFA can express that observing event *B* in state *s2* leads to state *s5* while observing *D* leads to *s3*. In a HMM, the states would not have this direct correspondence. For this reason, we decided to use the PFA as a probabilistic model.

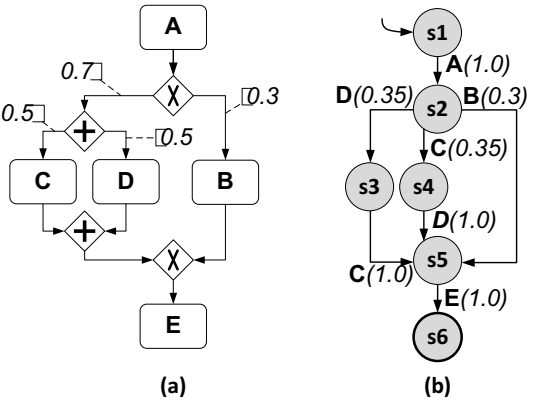

Figure 2: (a) shows a business process in BPMN notation with annotated probabilities. Annotations to the XOR-split represent probabilities of following the corresponding paths, while annotations to the AND-split represent probabilities of performing either *C* or *D* first. (b) shows a corresponding probabilistic automaton.

Numerous techniques for PFA learning have been developed in the literature. The authors of [VEH13] distinguish three types: state merging, Bayesian inference and parameter estimation. State merging techniques build a large PFA and iteratively merge states that are similar. Bayesian inference methods use sampling techniques to average over a large number of models. Parameter estimation uses maximum likelihood (ML) techniques to fit a standard PFA, usually with a fully connected state space.

In [VEH13], the authors also report results of a competition in which several methods have been benchmarked. The winning technique was based on Bayesian inference, directly followed by an ML parameter estimation approach. On first sight, these results suggest using Bayesian inference. However, prohibits model visualization as a huge number of models is drawn to average predictions over all of them. Since visualization is crucial to ensure comprehensibility, which is a key requirements of our approach, we resort to ML parameter estimation and defer Bayesian inference to future research.

ML estimation is prone to overfitting if the sample size is small. Small samples are considered a major challenge in working with event-based process data [AAM11]. Thus, precautions are necessary. We apply Bayesian regularization, which assumes that the

PFA's parameters are themselves random variables with a distribution [SJ02]. The standard choice for a PFA is the (symmetric) Dirichlet distribution, which is conjugate to the discrete distributions used in the remaining parts of the PFA [SJ02]. A user can modify the Dirichlet parameters to increase or decrease the tendency of the learning algorithm to output more evenly distributed parameters. Parameter estimation for models regularized in this way is called maximum a posteriori (MAP) estimation.

For PFAs, MAP estimation ca be done with the expectation maximization (EM) algorithm [DLR77], which was also used in [VEH13]. We have implemented a version of EM modified to work with a regularized PFA. We call this learning algorithm the *EmMapLerner*. A formal specification of the model can be found in equations (1) to (6). Equation (1) defines the distribution over the state in which the PFA is at the beginning of a process. Equation (2) defines the state-dependent distributions over the types of events observed at any point in time. Equation (3) defines the state- and event-dependent distributions over the states to which the PFA moves. Equation (4) to (6) define the corresponding Dirichlet distributions. K is the symbol denoting the number of states.

$$P(Z_0) \sim Discrete(\pi_0, \dots, \pi_K) \tag{1}$$

$$P(X_t | Z_t = k) \sim Discrete(b_{k0}, \dots, b_{kE}) \tag{2}$$

$$P(Z_t | Z_{t-1} = k, X_{t-1} = e) \sim Discrete(a_{ke0}, \dots, a_{keK}) \tag{3}$$

$$P(\pi_1, \dots, \pi_K) \sim Dirichlet(\rho_1, \dots, \rho_K) \tag{4}$$

$$P(b_{k1}, \dots, b_{kE}) \sim Dirichlet(s_{k1}, \dots, s_{kE}), \qquad \forall k \in \{1, \dots, K\} \tag{5}$$

$$P(a_{ke1}, \dots, a_{keK}) \sim Dirichlet(r_{ke1}, \dots, r_{keK}), \qquad \forall k \in \{1, \dots, K\}, e \in \{1, \dots, E\} \tag{6}$$

As K defines the dimension of the PFA, it is an input for EM. Determining appropriate values is known as the model selection problem [Mu12]. A standard approach is grid search, i.e., trying a range of values, fitting a model for each, and selecting the best of these candidate models [Mu12]. Applying grid search usually requires splitting the data into two separate parts, one to which the learning algorithm is applied, another to compare the candidate models. We have implemented grid search in the framework. It can also be used to systematically compare different degrees of regularization.

2.3 Model Transformation Algorithms and Visualizations

As the probabilistic model was chosen such that it can be interpreted as an automaton with a finite number of states and probabilities attached to the transitions, it is possible to visualize it graphically. For a given state k, b_{ke} is the (estimated) probability of observing an event of type e, and $a_{kej}$is the (estimated) probability of the PFA moving to state j if an event of type e is ovserved. Hence, $b_{ke}a_{kej}$ is the probability of the transition between the states k and j which is labeled with event type e.

Unfortunately, our EM learning algorithm estimates a transition probability for all the K^2E possible transitions of a fully connected PFA. A visualization will therefore be

unreadable unless the number of event types and states is very small. Thus, abstraction is necessary to generate a useful visualization. A simple but effective approach is to discard all transitions with a probability below a given threshold. This way, only the most important parts are visualized. This is the basic model transformation approach implemented in our framework and we call it *threshold pruning*. It delivers a visualization such as the one shown in figure 2 (b). Users can generate visualizations with varying thresholds to view the probabilistic model at varying levels of detail.

Even when using thresholds, the visualization can be unnecessarily complex. Different automata can be equivalent in terms of the behavior they describe, yet their structure can be different. Most importantly, the number of states can vary, which is why automata minimization is a problem with a long tradition in computer science [HMU01]. Different standard algorithms exist and have been evaluated in the BPM literature [BDD14]. They are all equally effective as they transform any automaton into an equivalent one with the minimum number of states. The Hopcroft algorithm however turned out to be most efficient in terms of runtime complexity [BDD14]. Hence, we integrated *automaton minimization* based on Hopcroft's algorithm into our framework.

Additionally, we provide a third transformation approach extending the other two. In BPM, higher-level notations such as BPMN, Event-driven Process Chains (EPC), and Petri nets are most popular [Aa13]. Thus, domain experts may refuse to use low-level automata-based visualizations. Fortunately, techniques exist that can transform automata to higher-level representations. We included Petrify [CKK97] into our framework, which is a tool used as a component of a process discovery algorithm [ARV10]. It transforms an automaton-based visualization into a Petri net. We call this model transformation approach *Petri net synthesis*. It is particularly useful for highly concurrent processes since petri nets can represent concurrency more compactly than automata.

2.4 Model Query Algorithms

The decision whether to use predictive models operationally is often not made by technical staff alone. Typically, at least one manager is involved. Managers often demand to understand the models at least at a basic level [PF13]. With our model transformation approaches, different visualizations can be produced that visualize the inner workings of a PFA. However, depending on the complexity of the event data and the visualization's level of detail, a visualization may just be too complex to be understood easily. Particularly because threshold pruning comes along with information loss, users may demand analysis on a low level of detail. A visual analysis could be tedious in these cases.

The problem of working with large sets of complex business process models is relevant in the BPM domain, which motivated scholars to develop numerous model-based analysis techniques [Aa13]. In particular, model query techniques allow specifying a pattern and then decide whether a process model contains this pattern. For instance, a pattern can be a partial specification of behaviour such as "event B can follow on event A" [Aa13]. Queries of these kinds applied to our visualizations can be used to verify that a probabilistic model actually behaves in a way that external domain knowledge suggests. It allows specifying and testing assumptions about the process.

Many model query algorithms can be found in the literature, either based on established theory such as temporal logic and graph theory, or based on custom-made algorithms (for surveys, see [Aa13], [BDD14], [DRR12], [WJW14]). As our primary objective is to include a generic query approach that could be applied to any type of model created by a model transformation approach, we decided to implement a graph-based approach first. A wide range of graphical visualizations can be interpreted mathematically as graphs, which consist of nodes and edges between the nodes. Both nodes and edges can be labelled to encode any information required for the matching. Our model query algorithm is based on an established algorithm for subgraph matching [CFS04], which was evaluated in the BPM domain and recommended as the most efficient algorithm [BDD14]. We call this model query algorithm the *subgraph pattern search*.

3 Demonstration and Evaluation

3.1 Data

To illustrate how our framework can be used and to evaluate how effective it is for predictive modelling of business processes, we apply it to real-world event data collected at Volvo IT Belgium [Do13]. The event data describes the incident and problem management processes. As process execution is supported with a dedicated information system, different status changes for registered incidents and problems are available.

The incident management event log consists of 3777 process instances with a total of 36730 events. Each event belongs to one of 11 types. The types describe the incident's status (a*ccepted*, *queued*, or *completed*) and substatus (*awaiting assignment*, *assigned*, *in progress*, *in call*, *resolved*, *closed*, *wait*, *wait-user*, *wait-implementation*, *wait-vendor*, or *wait-customer*). Similarly, the problem management event log consists of 744 process instances with a total of 4045 events. Each event belongs to one of 5 types which describe the status (a*ccepted*, *queued*, or *completed*) and substatus (*awaiting assignment*, *assigned*, *in progress*, *wait*, or *closed*) of the problem. This event log only contains data about problems that are already closed. To each process instance in both event logs, we added one event of a special event type called *END* to make process completion explicit. After this modification, the event logs had 40507 events of 12 different types and 4789 events of 6 different types respectively.

3.2 Analysis

To evaluate how effective the *EmMapLerner* is in building predictive models, we used it to implement predictors for different prediction problems. The first is predicting the type of the next event in a running process instance (or predicting that no more events will follow, i.e., that the process is finished). The second problem is focused on specific types of events. For each type of event and given a running process instance, the task is to decide if the next event will be of this type.

To build a predictor, we used the framework to fit a probabilistic model to both the incident and problem management event logs. We applied the *EmMapLearner* with the following parameters. As discussed in section 3.2, the number of states and the degree of regularization must be determined with grid search. For the number of states, we considered [2,4,6,8,10,12,14,16,18,20,30,40] as a set of candidate values. To express the degree of regularization, we make use of the interpretation of the (symmetric) Dirichlet distributions' parameters as pseudo-observations. This means that the degree can be specified relative to the number of events in the event logs [SJ02]. For instance, a degree of 0.1 can be interpreted as follows. For each of the event log's events providing evidence that any of the event type or state transition probabilities defined in equations (1) to (3) should be biased towards specific event types or transitions, 0.1 artificial events are added which provide evidence that all probabilities are equal. $[0.0, 0.1, 0.2, 0.3, 0.4, 0.5]$ is the set of candidate degrees we used. For each combination, we ran the EM algorithm. It is an iterative algorithm converging to a local optimum. We stopped iterating once the increase of data's log-likelihood was less than 0.001. The local optimum found with EM is not necessarily a global optimum, which is why EM is usually run many times with different initial values [Mu12]. We repeated EM 5 times per combination of state number and regularization degree with randomly generated initial values to reduce the risk of hitting bad local optima.

It is good practice to evaluate predictors on held-out test data not touched when fitting predictors [Mu12]. Moreover, grid search requires to split the event logs into one part used for EM and another part used for model selection. Hence, we spitted both event logs into three parts. 50% were used for EM, 25% to select the best model, and the remaining 25% were held-out test data to evaluate predictive performance. To measure performance, we generated all possible prefixes of all process instances in the test data. We recorded the accuracy with which the type of the next event was predicted (#correct predictions / # of predictions) as well as the sensitivities (true positives) and specitivities (true negatives) of the predictors deciding whether a given event type will be next. The results can be found in table 2. Sensitivities and specitivities have been averaged over all event types to present compact results.

To establish a benchmark for the performance of our approach, we implemented a simple prediction technique based on n-grams. It was also used as a benchmark in the grammatical inference competition [VEH13]. For each consecutive sequence $x_1, \dots, x_n$ of n evens, the next event's type is recorded. From the resulting frequency tables, probability distributions $P(x_{n+1}|x_1, \dots, x_n)$ are built which can be used for prediction. n must be chosen arbitrarily. We considered values from one to five. The results of these benchmark predictors can also be found in table 2.

The results demonstrate that no predictor can accurately predict what will happen next. However, this is not surprising since given a running instance, there is likely some degree of uncertainty regarding the type of the next event. However, the predictors are all much better than a predictor returning random guesses would be. For instance, with 12 (6) different types of events, the expected accuracy of random guessing is 0.083 (0.167). All accuracies of the predictors in table 2 are much higher.

Table 2: Results of the evaluation of the predictive power. Results of the *EmMapLearner* and the best of the predictors based on the last n events are boldface.

Process	Predictor	Accuracy	Ø Sensitivity	Ø Specitivity
Incident management	**EmMapLearner**	**0.714**	**0.383**	**0.974**
	Last1	0.621	0.346	0.965
	Last2	0.633	0.368	0.966
	Last3	**0.635**	**0.377**	**0.967**
	Last4	0.631	0.378	0.966
	Last5	0.624	0.375	0.966
Problem management	**EmMapLearner**	**0.686**	**0.519**	**0.944**
	Last1	0.690	0.521	0.945
	Last2	**0.699**	**0.564**	**0.948**
	Last3	0.686	0.553	0.946
	Last4	0.680	0.543	0.944
	Last5	0.665	0.530	0.942

The *EmMapLearner* performed better than the best of the benchmark predictors on the larger and more complex incident management event log. This is true for all three performance measures. However, the situation changes when considering the smaller problem management event log. While the predictor delivered by the *EmMapLearner* outperforms some of the benchmark predictors, it also is outperformed by others. These results suggest that building predictors with simple techniques such as using the last n events can be more effective for some datasets than using a more complex technique fitting a PFA. However, the PFA has its merits when processes are complex and context information not contained in the last n events is important.

To demonstrate how model transformation and model query analysis can be applied to gain insights about a PFA fitted with the *EmMapLearner*, consider the example in figure 3. On the left, the result of applying *threshold pruning* to a PFA fitted on the problem management event log is illustrated. The threshold was set such that only transitions with more than 15% probability are shown. Transitions are labelled with the corresponding event types and probabilities.

The visualization allows characterizing the predictive model's structure. The process most likely starts with an event of type *Accepted(in progress)*, which can lead to two different states. In the first, the process either proceeds directly with *Completed(closed)* or there is an event of type *Accepted(wait)* observed before completion. In the second state, the process proceeds with *Queued(awaiting assignment)* and then either returns back into the initial state with the same two event types observed so far or proceeds with *Accepted(in progress)*, *Accepted(assigned)*, *Accepted(in progress)*, and *Competed(closed)*. This description of the process can now be given to the users of the problem management system so that they can compare it to their experience with the system. It allows them to judge the appropriateness of the probabilistic model. It is important to note though that this visualization represents only the PFA's most likely transitions. Clearly, the process may behave differently in rare cases.

With a model query approach, judging the appropriateness of a probabilistic model could be done in a different way. It could start with defining patterns such as the two illustrated on the right of figure 3. The upper pattern shall specify a situation in which there are two transitions, one labeled *Accepted(in progress)* and the other labeled *Accepted(assigned)*. The probabilities do not matter, which is indicated with the symbol *. The dashed line from the second to the third state shall indicate that any path through the automaton should be a match in this query. From a system user, it could be known that whenever *Accepted(in progress)* is observed, it often happens that *Accepted(assigned)* will be observed later. To find out if the probabilistic model reflects this, a visualization could be queried with this pattern.

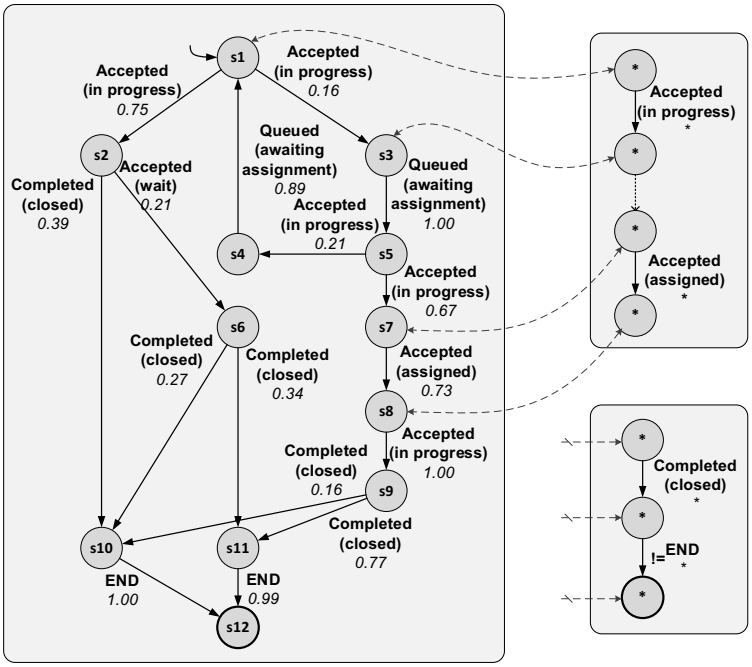

Figure 3: Visualization of a predictive model for the problem management process. Possible patterns for model queries can be found on the right.

Another possible pattern is illustrated on the lower right of figure 3. It matches if there is a transition labeled *Completed(closed)*, immediately followed by a transition labeled anything else than *END* (denoted with the *!=* modifier). It expresses a situation in which the process does not end after seeing the *Completed(closed)* event. Again, a model query algorithm can be used to ensure that there is actually no match for this pattern. It could be used even with very complex visualizations.

These two example patterns illustrate the limits of the *subgraph pattern search* algorithm, which is currently the only model query algorithm implemented in our framework. While the second pattern can be applied, the dashed line of the first pattern is a problem. Subgraph search does not support paths of arbitrary length. Hence, future versions of our framework require more extensive model query support.

4 Related Work

Predicting the future behaviour of running process instances is a topic in BPM research that has emerged only recently [APS10]. Thus, only few such approaches can be found in the literature. A notable example is presented by [ASS11], who use automata constructed with a process discovery technique to predict the time at which running process instances will be completed. Similar approaches for time prediction have been proposed as well (e.g., [RW13], [PNC11]). Our approach is different since its goal is to predict the events that are about to happen, not quantities such as time until completion.

There is also work on providing employees with recommendations. For instance, [SWD08] use an approach that compares running process instances with a set of (already finished) similar instances. Given a target to optimize, e.g., the total working time required for a process instance, the approach identifies the best action by predicting the outcomes of all actions available. Our approach is different since the goal is to predict what employees are likely to do, not what they should do to optimize a target function.

Similar to our approach is that of [LSD13]. Their goal is also to predict the event that will happen next, yet they consider different data. Their approach is tailored to case-oriented semi-structured business processes for which a large amount of context data is available. They characterize the current state of the process as all the data available and apply decision trees to predict the decisions employees will make. Another similar approach applying decision trees is presented in [Ro06]. Our approach is different since it assumes the existence of sequential event data.

In [RWA09], a simulation approach is presented which is designed to support operational decision making. By combining available process models from workflow management systems with operational event data, a simulation model is built and used to predict how running instances will behave. Our approach is different since it does not assume the existence of a process model.

Also related to our approach is the work of [Da98]. The author's goal is not to predict future events but to discover process models. However, the probabilistic technique they use is based on the same principles we applied in section 3.2 to construct the benchmark predictors. Effectively, the model discovery technique developed in [Da98] could be used to visualize the benchmark predictors in form of automata.

5 Current Status and Future Research

In this paper, we presented a framework designed to support predictive modelling of business process event data. We integrated techniques from different fields of research, most importantly grammatical inference, process mining, and conceptual model analysis. The learning algorithm has been evaluated on real-world event data and was compared against a suitable benchmark technique identified in the literature. Results indicate that the approach is effective. How to use the other parts of the framework has been illustrated with an example based on the data used in the evaluation. It uncovered a shortcoming

of the current model query component. Our framework is free to use and can be downloaded from https://github.com/DominicBreuker/RegPFA. We hope to encourage practitioners to use the approach and scholars to benchmark other approaches against it.

In the future, we plan to extend the framework in several areas. Since the evaluation of the *EmMapLearner* revealed that the benchmark predictors perform well and can even outperform our approach, including the benchmark predictors as learning algorithms is the next step. In [Da98], it is described how to construct a visualization using this approach, which is why it does not conflict with our requirement of easy visualization.

Furthermore, by applying the model transformation and model query approaches to the probabilistic model constructed in the experiments, we identified simple patterns that cannot be used with the model query algorithm that is currently implemented. Hence, a deeper analysis of the requirements will be conducted in order to prepare for a structured comparison of the more advanced model query approaches available in the literature.

References

[Aal11] van der Aalst, W. M. P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Berlin / Heidelberg: Springer, 2011.

[Aal13] van der Aalst, W. M. P.; Business Process Management: A Comprehensive Survey. ISRN Softw. Eng., vol. 2013, pp. 1–37, 2013.

[AAM11] van der Aalst, W. M. P.; Adriansyah, A.; de Medeiros, A. K. A.; Arcieri, F.; et al.: Process mining manifesto. Lect. notes Bus. Inf. Process., vol. 99, pp. 169–194, 2011.

[APS10] van der Aalst, W. M. P.; Pesic, M.; Song, M.: Beyond Process Mining: From the Past to Present and Future. 22nd International Conference, CAiSE 2010, 2010, pp. 38–52.

[ARV10] van der Aalst, W. M. P.; Rubin, V.; Verbeek, H. M. W.; van Dongen, B. F.; Kindler, E.; Günther, C. W.: Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. Softw. Syst. Model., vol. 9, no. 1, pp. 87–111, 2010.

[ARW07] van der Aalst, W. M. P.; Reijers, H. A.; Weijters, A.; van Dongen, B.; de Medeiros, A. K.; Song, M.; Verbeek, H. M. W.: Business process mining: An industrial application. Inf. Syst., vol. 32, no. 5, pp. 713–732, 2007.

[ASS11] van der Aalst, W. M. P.; Schonenberg, M. H.; Song, M.: Time prediction based on process mining. Inf. Syst. J., vol. 36, no. 2, pp. 450–475, 2011.

[BDD14] Breuker, D.; Delfmann, P.; Dietrich, H.; Steinhorst, M.: Graph theory and model collection management: conceptual framework and runtime analysis of selected graph algorithms. Inf. Syst. E-bus. Manag., 2014.

[CDN11] Chaudhuri, S.; Dayal, U.; Narasayya, V.: An overview of business intelligence technology. Commun. ACM, vol. 54, no. 8, pp. 88–98, 2011.

[CFS04] Cordella, L. P.; Foggia, P.; Sansone, C.; Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. IEEE Trans. Pattern Anal. Mach. Intell., vol. 26, no. 10, pp. 1367–72, 2004.

[CKK97] Cortadella, J.; Kishinevsky, M.; Kondratyev, A.; Lavagno, L.; Yakovlev, A.: Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. IEICE Trans. Inf. Syst., vol. E80-D, no. 3, pp. 315–325, 1997.

[CS12] Chen, H.; Storey, V. C.: Business Intelligence and Analytics: From Big Data to Big Impact. MISQ, vol. 36, no. 4, pp. 1165–1188, 2012.

[Da98] Datta, A.: Automating the discovery of AS-IS business process models: Probabilistic and algorithmic approaches. Inf. Syst. Res., vol. 9, no. 3, pp. 275–301, 1998.
[DLR77] Dempster, A.; Laird, N.; Rubin, D.: Maximum Likelihood from Incomplete Data via the EM Algorithm. J. R. Stat. Soc. Ser. B, vol. 39, no. 1, pp. 1–22, 1977.
[Do13] van Dongen, B.: BPI Challenge 2013. 2013. Retrieved from http://www.win.tue.nl/bpi/2013/challenge.
[DRR12] Dijkman, R.; la Rosa, M.; Reijers, H. A.: Managing Large Collections of Business Process Models - Current Techniques and Challenges. Comput. Ind., vol. 63, no. 2, pp. 91–97, 2012.
[EN10] Etzion, O. ; Niblett, P.: Event Processing in Action. Manning Publications Co., 2010.
[Hi10] de la Higuera, C.: Grammatical Inference. Cambride University Press, 2010.
[HMU01]Hopcroft, J. E.; Motwani, R.; Ullman, J. D.: Introduction to automata theory, languages, and computation. 2nd ed., Addison-Wesley Longman, 2001.
[JSC14] Jones, T.; Schulte, W.R.; Cantara, M.: Magic Quadrant for Intelligent Business Process Management Suites. 2014.
[LSD13] Lakshmanan, G. T.; Shamsi, D.; Doganata, Y. N.; Unuvar, M.; Khalaf, R.: A markov prediction model for data-driven semi-structured business processes. Knowl. Inf. Syst., 2013.
[Mu12] Murphy, K.: Machine Learning: A Probabilistic Perspective. The MIT Press, 2012.
[PF13] Provost, F.; Fawcett, T.: Data Science for Business: What you need to know about data mining and data-analytic thinking. O'Reilly Media Inc., 2013.
[PNC11] Pandey, S.; Nepal, S.; Chen, S. A Test-bed for the Evaluation of Business Process Prediction Techniques. 7th International Conference, CollaborateCom, pp. 382–391, 2011.
[Ra89] Rabiner, L. R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proc. IEEE, vol. 77, no. 2, pp. 257–286, 1989.
[Ro06] Rozinat, A.: Decision Mining in ProM. 4th International Conference, BPM 2006, 2006, pp. 420–425.
[RW13] Rogge-Solti, A. ; Weske, M.: Prediction of Remaining Service Execution Time Using Stochastic Petri Nets with Arbitrary Firing Delays. 11th International Conference, ICSOC 2013, 2013, pp. 389–403.
[RWA09]Rozinat, A.; Wynn, M. T.; van der Aalst, W. M. P.; ter Hofstede, A. H. M.; Fidge, C. J.: Workflow simulation for operational decision support. Data Knowl. Eng., vol. 68, no. 9, pp. 834–850, 2009.
[SJ02] Steck, H.; Jaakkola, T.: On the Dirichlet Prior and Bayesian Regularization. Neural Information Processing Systems, 2002, vol. 15, pp. 1441–1448.
[SK11] Shmuelli, G. ; Koppius, O. R.: Predictive Analytics in Information Systems Research. MISQ, vol. 35, no. 3, pp. 553–572, 2011.
[SMJ13] Schwegmann, B.; Matzner, M.; Janiesch, C.: A Method and Tool for Predictive Event-Driven Process Analytics. 11. Internationale Tagung Wirtschaftsinformatik (WI), pp. 721–735, 2013.
[SWD08]Schonenberg, H.; Weber, B.; van Dongen, B.: Supporting Flexible Processes through Recommendations Based on History. 6th International Conference, BPM 2008, 2008, pp. 51–66.
[VEH13] Verwer, S.; Eyraud, R.; de la Higuera, C.: PAutomaC: a PFA/HMM Learning Competition. Mach. Learn. J., 2013.
[VTH05] Vidal, E.; Thollard, F.; de la Higuera, C.; Casacuberta, F.; Carrasco, R. C.: Probabilistic finite-state machines - part I. IEEE Trans. Pattern Anal. Mach. Intell., vol. 27, no. 7, pp. 1013–1025, 2005.
[WJW14]Wang, J.; Jin, T.; Wong, R. K.; Wen, L.: Querying business process model repositories. World Wide Web, vol. 17, no. 3, pp. 427–454, 2014.