Business Processes Resource Management using Rewriting Logic and Deep-Learning-based Predictive Monitoring

Francisco Durán[a], Nicolás Pozas[a], Camilo Rocha[b]

[a] *ITIS Software, Universidad de Málaga, Málaga, Spain*
[b] *Electronics and Computer Science Department, Pontificia Universidad Javeriana, Cali, Colombia*

Abstract

A significant task in business process optimization is concerned with streamlining the allocation and sharing of resources. This paper presents an approach for analyzing business process provisioning under a resource prediction strategy based on deep learning. A timed and probabilistic rewrite theory specification formalizes the semantics of business processes. It is integrated with an external oracle in the form of a long short-term memory neural network that can be queried to predict how traces of the process may advance within a time frame. Comparison of execution time and resource occupancy under different parameters is included for several case studies, as well as details on the construction of the deep learning model and its integration with Maude.

1. Introduction

Business process optimization is the practice of increasing organizational efficiency by improving processes. The main motto of business process management in this area is to optimize processes to optimize business goals. Since efficiency is one of the major quantitative tools in industrial decision making, two of the most common goals are maximizing throughput and minimizing costs. For instance, companies using a business process over time can greatly benefit from streamlining a workflow, and by increasing the usage of resources within reasonable redundancy and costs limits. However, business process optimization is a multi-objective optimization problem in which many variables tend to be involved. One main challenge is to integrate predictive tools at design stages for devising business process optimization strategies.

Deep learning models are becoming increasingly important in business applications because they serve as a basis for monitoring and predicting process behavior (see, e.g., [9, 29, 35]). In particular, applications concerned with resource allocation, time and cost optimization, fault monitoring, and process discovery are using event logs to train these models and predict variables of interest. Logs usually contain information about processes in an organization as event lists. Each event is associated to a process instance, which is identified by a case number. A case is seen as a collection of activities, namely, tasks with attributes. Typically, they include a name or case number, a timestamp, and the resources or costs associated to it, among other attributes. Recurrent neural networks [46] have been widely used for performing sequence prediction. They are trained to learn from event logs and to predict sequences of 'next events' that are more likely to occur based on a partial, yet concurrent, process execution.

This paper proposes a two-layered approach to formal process optimization. The first layer uses the deep learning prediction power to help anticipate, given a partial execution, the demand of resources (i.e., number of replicas) in a business process. The second layer, built on top of the first one, formally specifies the concurrent behavior of process instances that compete for the same collection of resources and replicas. The outcome, i.e., the integration of these two layers, results in a sophisticated heuristic-oriented technique for the formal analysis and optimization of business processes. With its help, quantitative analysis can then be carried out by, e.g., computing the best combination of parameter values reducing costs, processing time, or resource waste.

Long short-term memory neural networks (LSTM) [46], a type of recurrent neural network, are used for sequence prediction as proposed in [34]. Given a business process B and a partial trace t of tasks in B, a LSTM $\mathcal{N}_B$ is used to predict an extension of t that conforms to B based on the (previous) training of $\mathcal{N}_B$ with event logs of B. A rewriting logic semantics of (a subset of) the Business Process Modeling and Notation (BPMN) simulates the concurrent behavior of instances of B, under a given set of constraints over the resources, by querying $\mathcal{N}_B$ as a scheduler and adjusting the number of replicas accordingly at runtime.

For training $\mathcal{N}_B$, the proposed approach takes as input the BPMN process B and a set of traces T_B of B. The process B has information about the type of resources needed

to complete the tasks. The traces T_B represent executions of B in, e.g., a production environment. The LSTM $\mathcal{N}_B$ is ultimately used to predict how resources are to be allocated/released in order to optimize their use; e.g., to minimize the time a resource is not being used or, similarly, to maximize the usage of resources meeting some budget and timing constraints associated to the process' execution. The LSTMs used in this work have been implemented in Python with Keras [6] and TensorFlow [2].

The integration of the prediction layer with Maude is met via socket communication: the rewriting logic semantics is the client of the prediction server run in Python. Given the partial concurrent execution of a number of B instances in the Maude semantics, $\mathcal{N}_B$ is queried with a time window. It returns a sequence of events extending the traces of the given instances, i.e., predicts continuations for the traces in that particular time window. These predictions are then used to adjust the number of replicas per resource at runtime in the Maude semantics executing B. The analysis is based on the simulation of the execution guided by such traces. The percentage usage of resources and the number of replicas during the time span of all replicas, among other, are monitored and summarized for each experiment.

Figure 1 depicts the logical design behind the proposed approach. This semantics has been designed and built to, e.g., read event logs, via sockets, keep the process' state as in an industrial system, and interact with prediction heuristics to improve resource allocation. The Maude process keeps a model of the system under control, and uses different strategies to make decisions on it. The focus of this paper, however, is on the predictive strategy to manage the allocation of resources. With this goal in mind, the system under control is simulated by a Maude process, also expressed using BPMN. This allows us to control the system under operation and compare different strategies on it. The strategy to automatically allocate/deallocate resources follows a design introduced in [17]. In the present work, neural networks are queried about which choices and durations are probable in the next steps of a process execution. This approach is presented on a guiding example in Section 3, with additional examples presented in Section 6. Section 6 also presents a comparison of the proposed strategy with other previously proposed strategies.

All the Maude specifications and Python code, together with examples, models and

Figure 1: Interaction of Maude between a neural network (left) and a BPMN process runtime (right).

outputs are available in a GitHub public repository [20].

This paper is an extended version of [10], with the following improvements:

- The LSTM model has been completely re-designed. Traces now contain additional information that allows a better prediction. A thourough analysis of the inputs and outputs to be consider has been carried out, and the model has been designed to minimize loss.
- New experiments have been designed, including new BPMN case studies for which different LSTM architectures have been explored and compared.
- The analysis of the experiments illustrating the proposed approach now includes a comparison with other non-AI prediction heuristics previously proposed by some of the authors.
- All sections have been rewritten, and the paper now includes a more self-contained treatment of topics such as LSTMs and the way they are being trained/used, and references have been added.

Outline. The rest of the paper is organized as follows. Section 2 gives some context to the current work by presenting some related and previous work. Sections 3 and 4 present overviews, respectively, of BPMN and LSTM neural networks. Section 5 presents the rewriting semantics of BPMN and details about how it interacts with the LSTM neural networks. Experimentation is presented in Section 6, and concluding remarks are gathered in Section 7.

2. Related Work

The results presented in this paper are part of a long-standing effort to provide BPMN modeling with extensions and formal analysis tools. It is based on rewriting logic [30], a computational logic that can naturally express concurrent computation.

In the broad spectrum of artificial intelligence, approaches based both on deep and machine learning have been used to tackle three classes of predictive business process monitoring problems related to business processes. Namely, next step prediction (e.g., the next task a process instance will likely execute), outcome prediction (e.g., how likely a client will file a complaint given a current instance state of a process), and quantitative process performance (e.g., the remaining execution time of each ongoing case of a process). In this paper, the proposed approach is focused on next-step and trace prediction to anticipate the availability of and need for resource replicas. As suggested by the literature [44, 23], deep learning (DL) techniques are employed; in particular, long short-term memory (LSTM) neural networks (NN) [46] are used for trace prediction. The reader is referred to [42] for an interdisciplinary comparison of sequence modeling methods for next-element prediction, including business process models and similar problems in other domains. Kratsch et al. [27] compare the performance of DL (i.e., simple feedforward deep NN and LSTM networks) and ML techniques (i.e., random forests and support vector machines) for outcome prediction in business processes. Moreover, Teinemaa et al. [43] and Verenich et al. [44] present surveys, respectively, on outcome-oriented and remaining time prediction methods in business process monitoring. Camargo et al. [4] compares various NN architectures to predict the activities, roles, and times of BPMN traces. In the present work, an architecture similar to the one named by them as full shared is implemented, where the NN is fed with information about the state of the resources used in the BPMN process to predict a next state. Camargo et al. [5] use full shared architectures to create generative NNs based on recurrent networks (GAN-LSTM), together with BPMN trace datasets, to create a simulator that is able to infer a complete BPMN process, including the generation of activities, start times, and duration of these activities. Hinkka et al. [22] use clustering-based pre-processing to simplify the input size of NNs, as well as to abstract certain attributes. They then add the label generated by these clusters as input to the NN. Given that predictive process monitoring has gained traction in companies, Di Francescomarino et al. [21] developed a value-driven framework for classifying existing work on predictive process monitoring. Their review can be used to support organizations navigate in the predictive process monitoring field by finding

value in the opportunities enabled by learning techniques.

In [36], Pika et al. identify several indicators on the behavior of resources, including skill, utilization, preferences, productivity, and collaboration, and extract patterns from real data. In [40, 8], the authors focus on the compliance of requirements related to data and resources. In particular, the A* algorithm is used in [8] to find the best process fitting a given event log. Huang et al. [24] use reinforcement-learning to assign resources with the goal of minimizing their long-term cost and to improve the performance of processes, in general. Nakatumba and van der Aalst [31] propose a method, based on regression, that provides the optimal balance between load and efficiency, thus establishing a relationship between load level and worker efficiency (when seen as resources). The approach of Tan and van der Aalst [41] and of Russell and van der Aalst [38] is similar to the one in [31], although in these two latter cases interaction between processes and humans is automated. Moreover, YAWL is used in [41], while BPEL4People [33] and WS-HumanTask [32] are used in [38] to define these interactions. In [39], Russell et al. characterize the lifecycle of resources and identify common workflow patterns. Finally, Yang et al. [47] propose a solution to learn execution contexts, with a focus on human resources and groups. For learning, decision trees are used to extract the categories of the rules given an event log; as output, they obtain simplified rules. The goal of the tree is to derive contexts, instead of building a classification or regression model.

On the basis of rewriting logic as a semantic framework, several techniques and approaches have been developed, and results have been presented. In [18], an encoding of timed business processes in the Maude language [7] has been proposed. It allows the automatic verification of several properties of interest on processes such as the maximum/minimum/average execution time. The same setting was used in [11] to analyze the timed degree of parallelism. In [12], stochastic specification of time and branching constructs, and statistical model checking are supported on a timed and probabilistic extension of BPMN. Duration times and delays for tasks and flows can be specified as stochastic expressions, while probabilities are associated to various forms of branching behavior in gateways. A probabilistic extension of rewriting logic (and Maude) and Maude's statistical model checker PVeStA [3] are used, respectively, for

discrete-event simulation and automatic stochastic verification of properties. They included expected processing time, expected synchronization time at merge gateways, and domain-specific quantitative assertions. Symbolic execution and verification was proposed in [13] with the help of rewriting modulo SMT [37]. Execution is driven by rewriting modulo axioms and by querying SMT decision procedures for data conditions. Reachability properties, such as deadlock freedom and detection of unreachable states with data exhibiting certain values, can be specified and automatically checked with the help of Maude, thanks to its support for rewriting modulo SMT. As part of this effort, analysis of resource allocation was proposed initially in [14] and then extended in [15]. Automatic computation of measures is enabled for precisely identifying and optimizing the allocation of resources in business processes, including resource usage over time. This approach has the advantage of capturing all concurrent behavior of processes and, thus, is used to simulate the concurrent evolution of any business process with a given number of resources and replicas.

More recently, the effort has shifted towards the optimization of processes by means of resource provisioning. In [16], an automatic analysis technique to evaluate and compare the execution time and resource occupancy of a business process, relative to a workload and a provisioning strategy, was proposed. Such analysis is performed on models conforming to an extension of BPMN with quantitative information, including resource availability and constraints. Four heuristics guided by, respectively, recent resource usage, recent resource request, predicted behavior, and a combination of available strategies are presented and compared in [17]. This approach is fully mechanized in Maude using bounded concurrent state-space search and waiting queues for tasks with shared resources. The reader is referred to the above-cited references for a more comprehensive summary of related work in each line of research.

3. The Business Process Modeling Notation (BPMN)

BPMN 2.0 is an ISO/IEC standard [25] that is extensively used for modeling business processes. BPMN is a graphical notation for modeling business processes as collections of related tasks that produce specific services or products. In BPMN, processes

are modeled using graphical representations for tasks and gateways, which are connected through flows and events. Although the specification of BPMN includes several diagrams, in this work, the focus is on the BPMN elements related to control-flow modeling and behavioral aspects. Therefore, two main kinds of BPMN diagrams are considered, namely, activity and collaboration diagrams. For these diagrams, the most common types of tasks, events, and gateways are included. Beyond those constructs, the BPMN specification is extended to also provide information on resources, as well as durations and delays of tasks and flows. This additional information will allow a real-time analysis of the specified processes.

The process in Figure 2 is used to introduce and illustrate the use of the supported BPMN constructs, and the analysis techniques presented in this work. Although intentionally simplistic, the process will help in guiding the presentation and discussion. Some additional examples will be presented in Section 6.

The workflow in Figure 2 models the on-line application for a visa, in which the user has to fulfil several forms, provide an electronic copy of his/her passport and pay the corresponding fees. The process consists of three lanes: one for the visa requester, one for the requirements checking, and one for the visa agency employee. The process starts with the client initiating the application by filling some basic information. A scanned version of the expired passport must then be provided. The size and quality of the uploaded file is checked. If the size of the uploaded file does not respect the size limit, or the quality is lower than required, the process terminates. If both the size limit and scan quality respect the imposed thresholds, the request is evaluated and a result is notified to the user (accept or reject). In case of acceptance, the user has to pay for fees and the bureau in charge of visa delivery prepares the requested document. Notice that both activities are achieved in parallel, since they are assumed independent. Finally, the visa is delivered.

The initiation and finalization of processes are represented by initial and final events. Events are also used to represent the sending and reception of of messages. Explicit message throw and catch intermediate events are used. A task represents an atomic activity that has exactly one incoming and one outgoing flow. A sequence flow describes two nodes executed one after the other, i.e., imposing an execution order between these

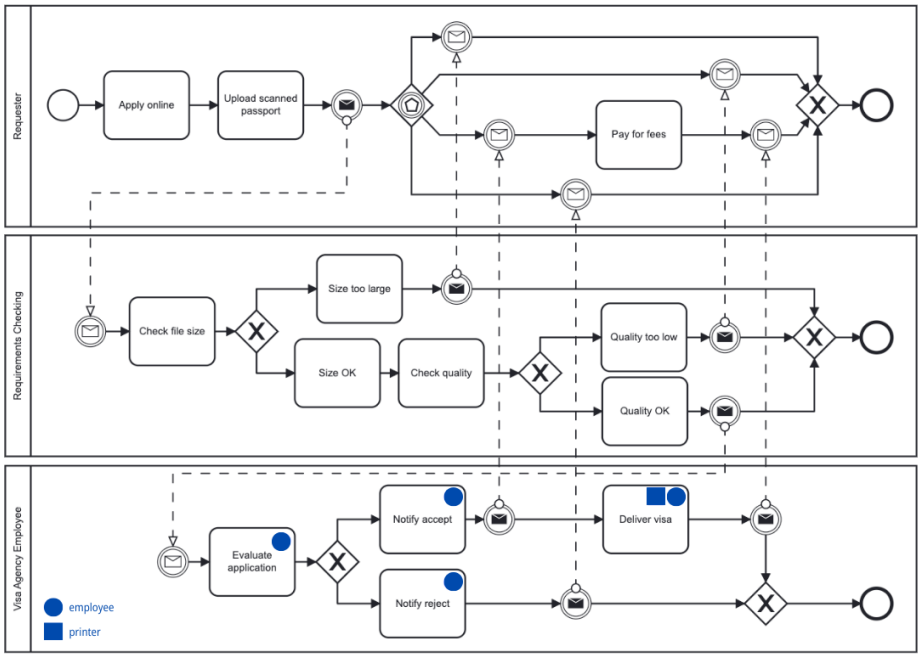

Figure 2: Running example: visa application.

nodes.

Gateways are used to control the divergence and convergence of the execution flows. In this work, *exclusive*, *parallel*, and *event-based* gateways are supported. Gateways with one incoming branch and multiple outgoing branches are called *splits* (e.g., split exclusive gateway). Gateways with one outgoing branch and multiple incoming branches are called *merges* (e.g., merge parallel gateway). An exclusive gateway chooses one out of a set of mutually exclusive alternative incoming or outgoing branches. A parallel gateway creates concurrent flows for all its outgoing branches or synchronizes concurrent flows for all its incoming branches. For an event-based gateway, it takes one of its outgoing branches or accepts one of its incoming branches based on events.

In addition to the description of specific tasks and their sequencing, collaboration diagrams also involve *pools* and *lanes*, which are structuring elements that split processes into pieces. In BPMN, each lane in a collaboration diagram corresponds to a specific role or resource. However, other resources may also be involved, and tasks

could require multiple resources or instances of the same resource. Therefore, instead of implicitly associating resources to lanes, resources are explicitly defined at the task level in the proposed approach. Hence, a task that requires resources for its execution can include, as part of its specification, the required resources. To depict it, symbols are associated to each resource type, and these symbols are included inside the corresponding tasks. Notice that resources can refer to humans (e.g., employee, cashier, executive), as well as non-human ones (e.g., robot, virtual machine, drone, tool). For example, the process in Figure 2 relies on employees for the evaluation and delivery of visas, and printers for the printing and delivery of the visas. For instance, the filled circles at the right-top corners of the Evaluate application, Notify accept, Notify reject, and Deliver visa tasks indicate that one instance of the employee resource is required for the execution of the tasks. Task Deliver visa requires instances of the employee and printer resources. In general, in addition to multiple resources, a number of them can be specified. For example, some weight of flour to prepare a recipe or some amount of money to purchase some product could be specified. To avoid dealing with multiple units of measurement, resources are counted as instances or replicas, and if more than one instance of a certain resource type is required, they are depicted as a number of icons in the task. The specification of resources may, nevertheless, include information, e.g., on the ranges to be consider—e.g., it may be the case in which more than five clerks in the company's premises cannot be allocated—or on the allocation time—a few days could be needed to allocate a new car, several minutes to allocate a virtual machine, or several weeks to hire a new engineer. In the visa example, it is assumed that a maximum of ten instances can be allocated for both employees and printers. Regarding allocation times, 10 employees and 1 printer are specified.

The process evolves by successively executing its tasks. However, the execution of a task requires the specified amounts of resources, which may lead to a competition for such resources: multiple instances of the process may also run concurrently and multiple tasks in the same run may require the same resources. In the running example, e.g., employees are used in several tasks and multiple persons may be trying to simultaneously apply for visas.

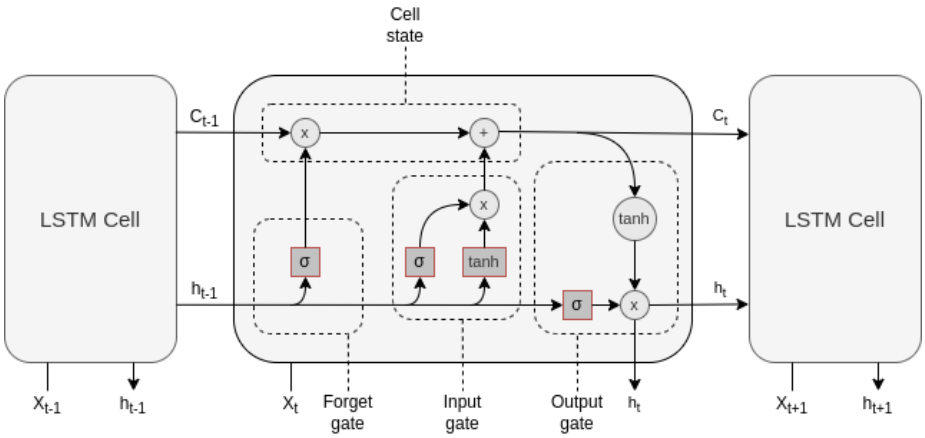

Figure 3: LSTM cell structure

4. Prediction of Event Traces with LSTMs

Recurrent neural networks (RNNs) are a type of neural network typically used to process and predict sequential data [46]. They consist of a set of neurons connected with each other, where each neuron has an input x_t and an output h_t at a specified time t, as well as a feedback loop that provides information of the previous timestamp. Long Short-Term Memory (LSTM) neural networks are a type of RNN used for sequence prediction. In particular, they are useful for solving problems involving long sequences that require information from the past. The architecture of this type of network includes feedback loops to maintain information over time and a set of gates that control the flow of information into a memory cell. The structure of LSTM memory cells is depicted in Figure 3. Memory cells enable the learning of long patterns using forget, input, and output gates.

4.1. Dataset pre-processing

RNN training requires a convenient data set, meaning that the data in this set must be well formatted and sufficiently varied, and irregular values such as, e.g., null values and NaN values must be corrected and converted to valid values (or eliminated if they are not necessary). All this data treatment is part of the pre-processing necessary and previous to model training. In the datasets used to train RNNs, special care must be taken to ensure that the sequences are valid and consistent within the dataset. Once the

data is prepared and ready to train a model, part of this data is typically used as *training data* (i.e., data with which the model must learn), part is used as *validation data* (i.e., data with which the model must correct learned errors), and part as *evaluation data* (i.e., data that the model has never seen previously and which is used to evaluate the behavior of the model).

4.1.1. Event logs

In this approach, RNN models of business processes are trained using execution event logs. These event logs are collections from timestamped events produced by the execution of business processes. For example, an event may specify that a given task started or completed at a given time. Event logs are used for different purposes, including process mining, conformance checking, etc. They may be represented using different formats, such as CSV, XES, MXML, XLSX, or Parquet. Independently of the format chosen, they typically include fields for date and time, an identifier, and a source. Nowadays, XES (eXtensible Event Stream) [1] is the most-widely-used standard for representing and storing event logs.

The core information for training consists of an event description, namely, an identifier (a number), an event identifier, and a timestamp:

- The identifier of the individual or process instance.
- The name of the event, which is unique, and may refer to a start or end event or the start or end of an activity.
- A timestamp defining the time at which the event has happened.

The following is a fragment of a sequence of events of the running example, where each line corresponds to an event in the execution of the process:

```
298, n004-init, 411.20
297, n014-init, 411.20
356, n004-end, 411.70
282, initial, 411.81
299, n004-end, 412.40
281, initial, 412.56
297, n014-end, 412.59
```

As it will be seen in the next sections, LSTM networks are quite good at predicting events (see, e.g., [21]). However, predicting the time at which these events will happen is a complete different issue. The prediction of time depends mainly on the problem

at hand and the current state of the system. For example, if there are no resource instances available for the execution of an activity, it will be postponed until enough resources become available. But this time may depend on the number of instances allocated, the current use of the resources, or the number of activities already waiting for them. In order to create useful models, networks need to be fed with additional information that can help it to make better predictions. In this case, traces are enriched with information on the state of use of the resources. Specifically, in addition to the identifier of the process instance, the event identifier, and the timestamp, each event includes information on the resources when the event occurs. For each resource, the following information is included:

- A name: the name of the resource.
- Its availability: the number of available instances of that resource.
- Its queue size: the number of pending requests for that resource.
- Its usage: the percentage of use of the resource.

This provides the model with additional information to more accurately predict what the next event will be, its duration, and the state of the resources in the next time period. For example, the following is a fragment of the trace for the visa example:

```
100, initial, 0, employee, 2, 0, 0.0, printer, 2, 0, 0.0
99, initial, 0.33, employee, 2, 0, 0.0, printer, 2, 0, 0.0
100, t1-init, 1.28, employee, 2, 0, 0.0, printer, 2, 0, 0.0
99, t1-init, 1.31, employee, 2, 0, 0.0, printer, 2, 0, 0.0
100, t1-end, 4.10, employee, 2, 0, 0.0, printer, 2, 0, 0.0
99, t1-end, 4.12, employee, 2, 0, 0.0, printer, 2, 0, 0.0
98, initial, 4.49, employee, 2, 0, 0.0, printer, 2, 0, 0.0
100, t2-init, 4.98, employee, 2, 0, 0.0, printer, 2, 0, 0.0
97, initial, 5.04, employee, 2, 0, 0.0, printer, 2, 0, 0.0
99, t2-init, 5.13, employee, 2, 0, 0.0, printer, 2, 0, 0.0
```

4.1.2. Data pre-processing

Even though the data used to train the network comes from event traces, they cannot be fed to the network as such, but must be processed before initiating the training itself. For instance, during the processing of the traces, event identifiers are collected. For efficiency, an integer value in the range $[0..n]$, with n the number of events, is associated to each of these event identifiers, which are then used as indexes for different maps and hash tables. In order to predict the next activity and the estimation of resource usage,

the activities in the dataset are sorted by use case and by timestamp. Once this is done, each activity is enriched with the information about its next activity.

Another important issue related to the prediction of the time at which a given event will happen is that such a time does not only depend on the state of the process, but also on other factors. They include the time at which each specific process instance initiated its execution, the number of instances being run concurrently, and if multiple tasks inside the same instance may be executed concurrently. Therefore, instead of predicting the timestamp of an event, the focus is on the lapse of time between events of the same instance. In the case of an activity-init event, the lapse of time until the corresponding activity-end is predicted, i.e., the duration of such an activity. In the case of an activity-end, a start, or an end event, the goal is to find the time until the initiation of the following activity or the occurrence of an end event, i.e., the delay in the flows and intermediate gateways. Therefore, instead of training the network with the raw timestamps, the differences are calculated as explained above. That is, if activity *t1-init* has a timestamp 2 and activity *t1-end* has a timestamp 5, the number associated to *t1-init* is 3 time units. Trained in this way, the network will be able to predict durations and delays, which will later be used to create the corresponding timestamps.

4.1.3. Most-informative variables

Once the initial pre-processing is done, the data is organized to be fed to the network for training. Output variables are identified, as well as the most informative input variables for each of them and how long the sequences of these variables are going to be.

A correlation matrix is used to analyze how the variables are related to each other. A correlation matrix indicates, by means of a Pearson coefficient (i.e., a numerical value in the range $[-1, 1]$), how related two variables are. With the help of such a matrix, input variables most closely related to the target variables have been identified:

- The target activity strongly depends on the previous activity, and on the queues and usage percentages of resources.
- The duration strongly depends on the sizes of the resource queues.
- Queue sizes and resource usages strongly depend on the previous activity, previ-

ous usage of resources, and process identifier.

With this information, a sub-sequence size of events is defined to build the dataset for the training process. Indeed, the choice of this size is an important decision. If a sequence size too small is chosen, the model may not identify loops and then make wrong predictions.

4.1.4. Sub-sequence sizes

In the long run, with appropriate resource allocation strategies, any process should end up stabilizing its execution. However, when the conditions change, the number of instances of the resources must be updated until a stable situation is recovered. For training, having long traces is not as important as having a significant heterogeneous wealth of different traces in different circumstances.

Instead of arbitrarily choosing a sequence length, different tests were carried out to find out the best value. The main criteria is to minimize *loss*. Roughly, the loss function is the goal function. In neural networks, the goal is to minimize error and the term loss is used to refer to the error. This loss function is typically used as a guide in validation since the objective should be to minimize this function, meaning that the model has the smallest possible error in data with which it has not been trained. In summary, the goal is to find a sequence size that minimizes the loss.

This length depends on the particular problem at hand. The table in Figure 4 depicts the loss value for the visa process (see Section 3) for different validation event logs, of different sizes. Specifically, the table shows the error value when training the network with different sequence lengths along successive epochs. In NN terminology, an epoch is the process of training the network with all the training data available. The network will be trained by repeatedly entering this data until a stop criteria is reached. Each of the rows in the table shows the loss value (Value) for the corresponding sequence length, the number of epochs after which the training stops (Steps), and the execution time of the training (Time).

The sequence length is chosen by looking at the loss value, but the training time and other parameters may also be taken into account. In the info in Figure 4 it can be seen that the loss times are quite similar for the different lengths, with length 25 being

Length	Value	Steps	Time
05	6.28	18	0h 40m 57s
10	6.13	25	1h 02m 29s
15	6.02	35	1h 33m 08s
20	6.17	22	1h 04m 04s
25	5.93	44	2h 17m 41s
30	6.14	23	1h 20m 26s

Figure 4: Visa model training with different sequence lengths.

the best one. However, its training time is 2h 17m, whilst other lengths with close loss values show a considerable smaller execution time. In this case, the choice is 15, which has a value of 6.02 and training time of 1h 33m.

Traces are grouped in batches. These batches have a fixed size. In this case, batches have default size of 32.

Once the data has been processed and analyzed, the dataset to train the model can be created. This data is split into the input dataset (X) and output dataset (Y). In this case, three different outputs from the same model are used: the first output classifies the target activity, the second one performs a regression for the duration, and the last one performs a regression for each piece of information on the resources.

4.2. The model

Figure 5 shows a graphical representation of the model. Each of the boxes in the figure represents a layer and the arrows the direction in which they are interconnected. The network has been built using TensorFlow [2] and the used layers are instances of predefined layers available in the TensorFlow platform. The type of a layer is indicated in the bottom-left slot of its layer box. The purpose of each of the used predefined layers is the following:

- An InputLayer defines the way in which the data is collected, it does not apply any modification to the data.
- A Concatenate layer just concatenates the received input layer data.
- A BatchNormalization layer applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.
- An LSTM is a Long Short-Term Memory layer.
- A Dense layer is a regular densely-connected NN layer, where all neurons are

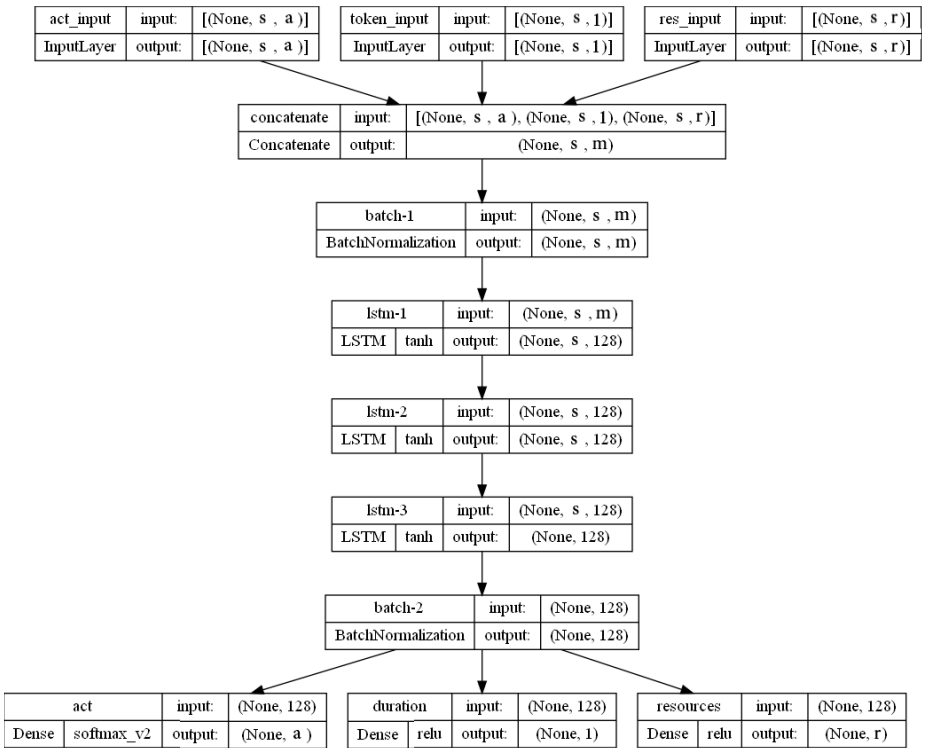

Figure 5: LSTM model structure.

connected to all inputs.

In addition to the identifier of the layer, given at the top-left corner of each box, the rest of the information describing each layer corresponds to its input and output. Datasets have the form (X, Y, Z), where X represents the batch size, Y is the size of each sequence, and Z the size of the attributes. The Python keyword *None* is used as default value. As it will be seen below, the number of attributes depends on the specific business process.

At the top of Figure 5, there is the input layer with three inputs: *act_input* for the activities, *token_input* for the execution instance, and *res_input* for the resources. Activities are encoded using the OHE encoding.[1] Thus, the *act_input* layer has a default batch size, its sequences have length s, and has a different activities. The second input corresponds to the process execution identifier. It is a number identifying a token or individual of the current population. The third input, *res_input*, corresponds to the resources handled by the process. For each resource, it includes the number of instances of that resource type available, the number of instances in the pending queue, and the percentage of use of that resource. This results in an input of length three times the number of instances of the specific process.

After the input layer, there is a concatenation layer that converts the different inputs into a single input. In the concatenation layer, $m = a + 1 + r$. The input of the concatenation layer is then normalized by a batch normalization layer to ensure that each of the features will have the same relevance in the model. Subsequently, three LSTM layers of 128 units are used, with dropouts [0, 0.2, 0]—see the discussion below on why these numbers. The first two LSTM layers return the complete state to the next layer to maintain all the information, while the last layer only returns the information of the units. Relating Figure 5 and the cell structure in Figure 3, the first two layers

[1] OHE stands for One-Hot Encoder. These networks take integer arrays as inputs, and the OHE encoding allows to include categorical values, such as strings and discrete numbers, in a sparse matrix. The features are encoded using a one-hot encoding scheme. To do this, the number of unique elements are counted and a vector with such a length is created. This vector has a 0 in all its positions, except for the position corresponding to the discrete value. For example, if there are 6 different categories, the encoding of category 2 would give the result [0, 0, 1, 0, 0, 0]. This creates a binary column for each category and returns a sparse matrix.

output C_t and h_t, while the last one only outputs h_t. A normalization layer has been applied again to ensure equality weights between features. Finally, the three outputs are defined using this last normalization layer.

Dense layers with Softmax and ReLU activation functions are used to define the outputs. The target activity is defined using a Softmax activation function, and duration and resources are defined using the ReLU activation function. The *Softmax* function converts a vector of values into a probability distribution, where the elements of the output vector are in the range (0, 1) and sum up to 1. The *Softmax* function allows, given a set of categories (in this case the different events), to estimate the probability of occurrence of each of these categories. The *ReLU* function is a rectifier that avoids negative numbers, so the output value of this function is the maximum between 0 and the input value. In this way, it avoids making predictions with negative value in the duration or in the information about the resources.

This section concludes with a discussion regarding the election of the number of layers, the number of neurons per layer, and the values of the dropouts of the different layers. To find out the best possible structure and values, the implementation of the Hyperband algorithm [28] of the KerasTuner library[2] was used for the visa process, and also for the other case studies in Section 6.4. The conclusion is that the best configuration corresponds to the following parameters:

- Number of LSTM layers: range [0, 5].[3]
- Number of neurons per layer: 64, 128, and 256 are used as alternatives.[4]
- Dropout per layer: 0.0, 0.2, or 0.4 are used as an alternative values.[5]
- Batch normalization between layers: Keras indicates if a normalization layer is

[2]The KerasTuner library: `https://keras.io/keras_tuner/`.

[3]An extra layer must be added to the result, since there must be at least one LSTM layer, the last one, which must make a flattening of the input sequences to pass it to the output layers.

[4]The number of neurons in each layer must be greater than or equal to the number of neurons in the following layer. That is, if 128 neurons are chosen in the first layer, then 256 neurons cannot be used in the second layer. Therefore, in the last layer it must be chosen between the number of neurons of the previous layer, and the maximum between the number of neurons of the previous layer divided by two and the number of features (activities in this case).

[5]The dropout is an effective and simple technique to reduce overfitting in training. Basically, it is the probability that a neuron is deactivated, making the weight that should be carried by that neuron be carried by another neuron, and thus creating redundant neurons that are able to react to certain patterns.

useful or not.

4.3. Training results

Once the data has been prepared to train the LSTM model, 70% of the available traces are used as training set and the remaining 30% as the testing set to validate the accuracy of the model. The evaluation is then done directly with the predictor integrated with the Maude-based allocation strategy (see Section 5.3). In each prediction, the LSTM model assigns a probability to all the tasks to decide which one will happen next. Furthermore, the prediction is adapted to take as input a number of tasks to predict, which allows prediction on different lengths in Section 6.

The quality of the training is evaluated by using the crossentropy loss and the mean squared error (MSE). For the optimizer, Adam (adaptive moment estimation) [26] has been used with an initial learning rate of 10^{-4} that will be decreasing with a factor $lr = lr \times 0.2$ if no improvement is achieved in five epochs. The TensorFlow accuracy metrics used for activities are: (1) categorical accuracy, which calculates how often predictions match one-hot labels; (2) precision, which calculates the precision of the predictions with respect to the labels; (3) recall, which calculates the recall of the predictions with respect to the labels; and (4) AUC, which calculates the area under the ROC or PR curves—the receiver operating characteristic (ROC) curve and the precision-recall (PR) curve are two visual tools for comparing classifiers.[6]

No accuracy metric is used for duration and resources, instead the MSE value is used directly. Training is terminated when 100 epochs are reached or no improvement in loss value is achieved for 8 epochs. The training of the visa process shows a loss value of 5.57, which is reached after 86 epochs. The training in this case took 5h 5m 40s.

5. Rewriting Logic Semantics and its Integration with the LSTM Model

As explained in Section 1, BPMN processes are modelled in Maude with two different goals. In the first case, the execution of a process is guided by an event log. In

[6]Information on the accuracy metrics provided by Keras is available on line at `https://keras.io/api/metrics/accuracy_metrics/`.

the second case, the process is *autonomous*, in the sense that its execution proceeds randomly, taking into account stochastic expressions that model the different decisions that happen: decisions on which branches to take, durations of activities, and delays in flows. This information on the behavior of the system might be provided by real experts or learned from the same event logs. In this section, computational simulations are used to represent many possible behaviors under this autonomy. In this section, these two alternative ways of evolving business process models are presented.

The autonomous process is responsible for the simulation of the system and the managing and analysis of resources. It is also in charge of communicating with the Python process performing the predictions. The Maude process submits the event log of the activities carried out and, periodically, requests predictions. At this time, the Maude BPMN process creates a replica of itself, which will be guided by the event sequence submitted by the predictor. Once the guidance consumes all the predicted events, the status of the resources is analyzed to decide on the allocation/releasing of replicas, and the simulation is restored. In summary, sharing a common representation of the core elements of processes, one specification defines the evolution of the process in accordance to the events in the log trace, whilst the other one non-deterministically progresses using the information with which models are annotated.

Section 5.1 presents autonomous processes and the additional information required for its execution. How the modeling of event-based processes is realized is explained in Section 5.2. The machinery for using the predictions to guide the execution of the process, and the analysis of such guided execution to update the allocation of resources, is explained in Section 5.3. The specification builds on a specification that has evolved along different extensions through time [18, 11, 12, 13, 14, 15, 16, 17].

5.1. Autonomous processes execution

Although both the specification of autonomous processes and of event-guided processes share a significant part of the structure, the specification of the autonomous process requires additional information. To simulate the execution of processes, the process specifications are enriched with some quantitative information that is used to make the required decisions. This additional information is added as annotations to

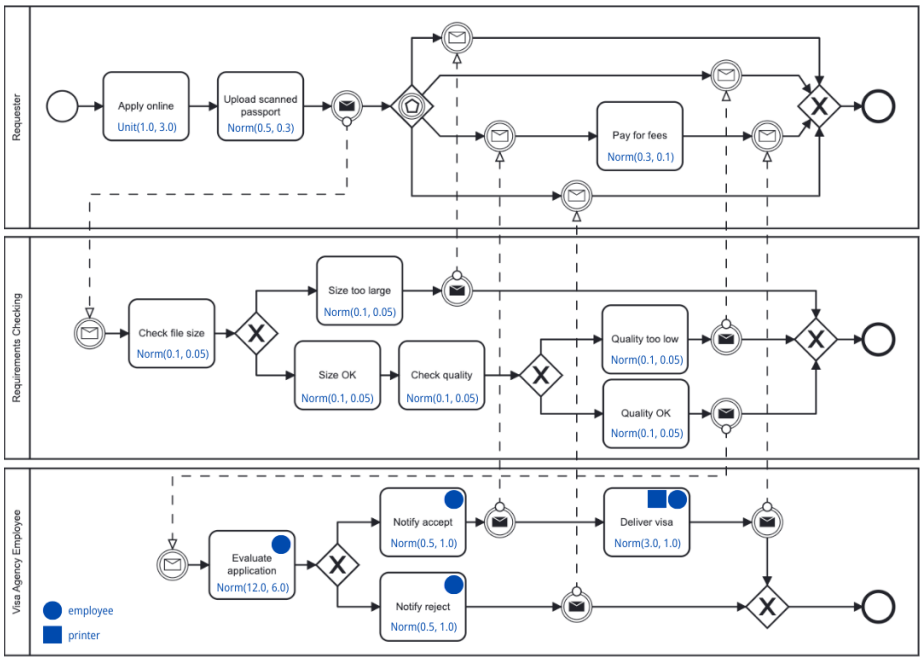

Figure 6: Running example: visa process with resources, durations and probabilities.

the process model. Specifically, duration and delay associated to tasks and flows are expressed as stochastic expressions. Similarly, alternative constructs (split exclusive gateways) are extended with probabilities associated to outgoing flows. Figure 6 shows the process given in Figure 2 enriched with such information.

5.1.1. Additional information for autonomous execution

Data-based conditions for split gateways are modeled using probabilities associated to outgoing flows of exclusive and inclusive split gateways. For instance, notice the exclusive split after the Check file size task in the Requirements checking lane of the running example, which has outgoing branches with probabilities 0.2 and 0.8, specifying the likelihood of following each corresponding path. The probabilities of the outgoing flows in an exclusive split must sum up to 1.

The timing information associated to tasks and flows (durations or delays) is described either as a literal value (a non-negative real number) or sampled from a probability distribution function according to some meaningful parameters. The probability distribution functions currently available include exponential, normal/Gauss, and uni-

form (see, e.g., [45]). In Figure 6, the specification of task duration has been placed at the bottom of each task. In the modeling tool, these parameters would be specified as properties of the corresponding element. For instance, the duration of the Apply online task is specified as Unif(1.0, 3.0), which means that it follows a uniform distribution in the range $[1, 3]$. The Upload scanned passport task follows a normal distribution with mean 0.5 and variance 0.3, that is specified as Norm(0.5, 0.3). To simplify the presentation of the running process, the delays in all flows are set to Norm(1.0, 0.2) to express that they all have a delay that follows a normal distribution with given parameters.

5.1.2. The specification of BPMN processes

In the Maude specification of BPMN, a process is represented as an object with sets of nodes and flows as attributes. The representation of each node type includes the necessary information to describe its structure and to contribute to the overall process analysis. Given unique identifiers for nodes, flows, resources, and events the process of the running example can be specified as shown in the excerpt in Figure 7. Figure 8 adds to Figure 6 the identifiers used in the Maude representation.

The specification in Figure 7 shows how a Process object has attributes with the definition of its nodes and flows connecting them. The start event is specified in line 3, it includes an identifier, "initial", and an output flow, "r1". A task node involves an identifier, a description, two flow identifiers (input and output), a stochastic function modeling its duration, a set of resources required for its execution, and a set of messages to be delivered after its completion. For example, the apply online task, specified in line 7, has identifier "t1", and "r1" and "r2" as input and output flows, respectively; its duration follows a uniform distribution in the range $[1, 3]$, does not send any message, and does not require any resource. Flows "r1" and "r2" are specified in lines 14–15. The representation of a flow includes a probability distribution function specifying its delay, and an optional message or timer. The message blocks the flow until it is received, whereas the timer represents a delay after which the execution is triggered. A split node includes a node identifier, a gateway type (exclusive, parallel, or event-based), an input flow identifier, and a set of output flow identifiers. For example, the exclusive split "g4" (line 6) has "c2" as incoming flow, and "c3" and "c4", with associated probabilities 0.2

```
1 < pid : Process |
2     nodes :
3       (start("initial", "r1"),
4        merge("g6", exclusive, ("c5", "c10", "c11"), "c12"),
5        split("g2", eventbased, "r4", ("r5", "r6", "r7", "r8")),
6        split("g4", exclusive, "c2", (("c3", 0.2) ("c4", 0.8))),
7        task("t1", "apply online", "r1", "r2", Unif(1.0, 3.0), empty, empty),
8        task("t10", "evaluate application", "e1", "e2", Norm(12.0, 6.0),
9          ("employee"), empty),
10       task("t13", "deliver visa", "e5", "e7", Norm(3.0, 1.0),
11         ("employee", "printer"), ("mdeliver")),
12       ...),
13    flows :
14      (flow("r1", Norm(1.0, 0.2)),
15       flow("r2", Norm(1.0, 0.2)),
16       flow(id("r5"), Norm(1.0, 0.2), message("msizebig", "size too big")),
17       flow(id("r6"), Norm(1.0, 0.2), message("mqualitylow", "quality too low")),
18       flow(id("r7"), Norm(1.0, 0.2), message("maccept", "accept")),
19       flow(id("r8"), Norm(1.0, 0.2), message("mreject", "reject")),
20       flow("r11", message("cinitiated", "checking initiated"), "c1"),
21       ...) >
```

Figure 7: Running example: Maude representation of the visa process.

and 0.8, respectively, as outgoing flows. The event-based split gate "g2" (line 5) has "r4" as incoming flow, and "r5", "r6", "r7", and "r8" as outgoing flows. Note the definition of these flows in lines 16–19; after the corresponding delay, they become active upon the reception of the corresponding messages. A merge node includes a node identifier, a gateway type, a set of input flow identifiers, and an output flow identifier. For example, the exclusive merge "g6" is specified in line 4.

5.1.3. The dynamics of autonomous processes

A set of rewrite rules specifies how *tokens* evolve through a process. Each move of a token inside a BPMN process is modeled as a rewrite rule. For example, one of the actions that may occur, and that is modeled by a corresponding rewrite rule, is that when there is a token in the incoming flow of an exclusive split, the token is moved to one of the outgoing flows of the gate, with its timer set to the value resulting from evaluating the stochastic expression of the flow, which represents the delay of the flow. Objects of classes Simulation, Workload, and Supervisor manage different aspects of the simulations. The main aspects of these classes and other elements of the specification are detailed next.

Simulation. While process objects represent static processes and they do not change along simulations, all the information on process execution is kept in simulation objects. Specifically, a Simulation object—see Figure 9—stores a collection of tokens (in

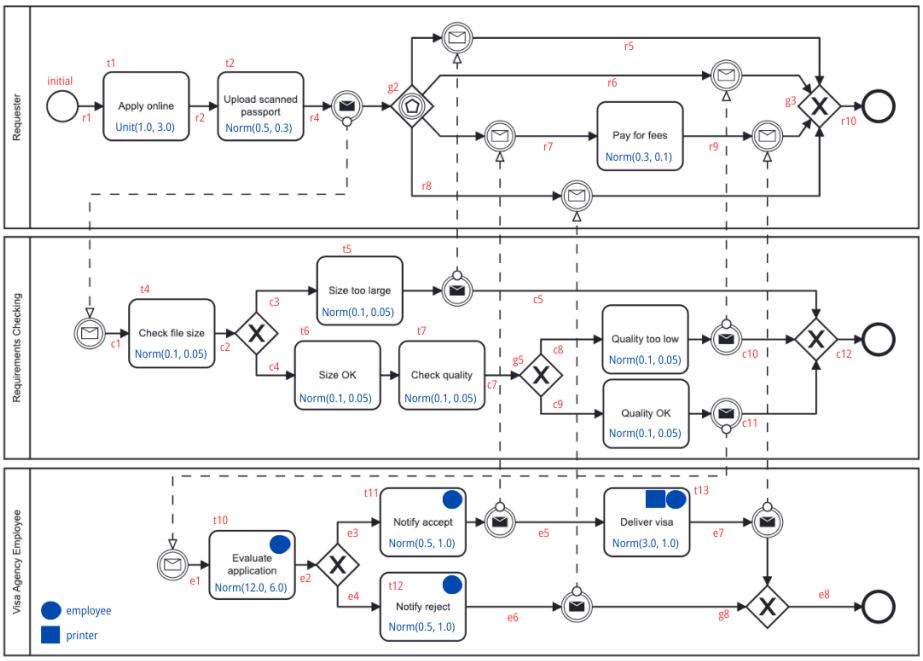

Figure 8: Running example: visa process with identifiers of nodes and flows.

```
1 class Simulation |
2   tokens : List{Token},                ---- scheduler
3   gtime : Time,                        ---- global time
4   resources : Set{Resource},           ---- resources in the system
5   events : Map{Id, Set{Event}},        ---- events in each execution
6   process-execs : Map{Id, Time},       ---- execution time of each execution
7   sync-times : Map{Id, Map{Id, Time}}, ---- sync. time of each gate in each execution
8   task-times : Map{Id, Map{Id, Time}}, ---- task execution times
9   ...
```

Figure 9: Declaration of the Simulation class (partial, please, note the ellipsis).

a tokens attribute), a global time (gtime), a set of events (events, including messages and timers), and a set of resources (resources). It also keeps track of the metrics being computed. For analysis purposes, during the execution of a process, some information is collected in the corresponding attributes: time stamps, task durations, and waiting time at parallel gateways. This information is necessary for guiding the execution of the process, periodically evaluating the amount of resource instances, and for presenting the results to the user for possible optimizations.

Tokens. Tokens are used to represent the evolution of the workflow under execution. Since there may be several simultaneous executions of a process, each execution is identified with a unique identifier, which is used to associate tokens to executions. Thus, a token is represented as a term token(TId, Id, T), where TId is the execution instance the token belongs to, Id is the identifier of the flow or node it is attached to, and T represents a timer, of sort Time, modeling a delay of the token, which represents the duration of a task or the delay associated to a flow. Once its timer becomes 0, a token can be consumed.

Tokens are stored in the tokens attribute of the Simulation object—implemented as a priority queue, so that tokens are processed according to their due time. However, even if a token is at the front of the queue with timer 0, its execution may be delayed. For example, consider a task that requires a resource that is not available or a parallel merge for which some incoming flow is not yet active. To avoid deadlocks, the scheduler implements a *shifting* mechanism that identifies the first active token to the front of the queue in case the current head needs to be delayed.

Events. A message event may be associated to a flow, which is blocked until the message is received. A timer event may also be associated to a flow. When a token arrives at a timer event, its countdown starts: once the countdown is completed, the token moves to the outgoing flow. Both message and timer events are usually associated to event-based gateways, but this is not always the case (see, e.g., the initial flow for the Requirements Checking or Visa Agency Employee lanes in the process in Figure 2, 6, or 8). Asynchronous events are modeled using a map that associates an event set to each process execution instance in the Simulation object: when a message is dispatched, a corresponding event is added to the corresponding set. Flows and gateways waiting for specific messages use these sets to check whether messages have arrived.

Resources. For each resource type, a number of instances or replicas are provisioned. At each moment during a simulation, some of these instances can be in use and others can be available for tasks to use them. Section 5.3 explains how the supervisor scheme presented in [17] has been used for the specification of different strategies for the dynamic provisioning and releasing of resource instances along executions, and how it

```
1 sort Resource .
2 op resource :
3      Id                          ---- identifier of the resource
4      Nat Nat                     ---- range on the number of resources (min, max)
5      Time                        ---- allocation time
6      Nat                         ---- total number of replicas
7      List{2-Tuple{Float, Nat}}   ---- total number of replicas along time
8      Nat                         ---- number of available replicas
9      List{2-Tuple{Float, Nat}}   ---- number of available replicas along time
10     Bag{Replica}                ---- replicas of the resource
11     Nat                         ---- size of the resource's queue
12     List{2-Tuple{Float, Nat}}   ---- evolution of the size of the queue along time
13     List{2-Tuple{Float, Float}} ---- resource usage along time
14     Time                  ---- time all replicas are in use (reset at supervisor's checks)
15     -> Resource [ctor] .
```

Figure 10: Definition of resources.

has been extended for the definition of the resource allocation strategy based on NN predictions.

Given a number of provisioned resources, if a running task requires resources and they are available, it blocks them and initiates execution immediately. Indeed, whenever a task requires instances of several resource types, it *atomically* picks them or waits for all of them to be available. If the required resource instances are not all available, resource requests are submitted and the task remains blocked until its requests are satisfied. To support this, each resource type keeps a queue of requests.

Eventually, new resources may be provisioned, i.e., added to the pool of available resources or released (i.e., removed from such pool). Resource provisioning is not instantaneous, nevertheless. There might be some delay in the allocation of new instances; this time is referred to as *allocation time* (AT).

Each resource type is represented by a resource operator that gathers all required information: an identifier, the minimum and maximum number of allocatable replicas (0, if unlimited), its allocation time, the total number of allocated replicas, the number of available replicas, the total amount of time the replicas of this resource type have been in use, and some historical information on resource usage, request queues, etc., which are handy for analysis purposes. Figure 10 shows the specification of this class.

Tasks. The execution of a task is modeled with two rules. The first rule, the initTask rule shown in Figure 11, represents the task initiation, which is applied when a token with zero time is available at the incoming flow (line 5). If all the resources required

```
 1 rl [initTask] :
 2    < PId : Process |
 3       nodes : (task(NId, TaskName, FId1, FId2, SE, RIds, SEI), Nodes), Atts >
 4    < SId : Simulation |
 5       tokens : (token(TId, FId1, 0) Tks),
 6       task-tstamps : TTSs, gtime : T, resources : Rs, Atts1 >
 7    < CId : Counter | counter : N >
 8 => if allResourcesAvailable(RIds, Rs)
 9    then < PId : Process |
10           nodes : (task(NId, TaskName, FId1, FId2, SE, RIds, SEI), Nodes), Atts >
11         < SId : Simulation |
12           tokens : insert(Tks, token(TId, NId, time(eval(SE, N)))),
13           task-tstamps : if TTSs[TId][NId] == undefined
14                          then insert(TId, insert(NId, T, TTSs[TId]), TTSs)
15                          else TTSs
16                          fi,               ---- for loops, stamps get overwritten
17           gtime : T,
18           resources : grabResources(RIds, Rs, time(eval(SE, N)), T), Atts1 >
19         < CId : Counter | counter : int(eval(SE, N)) >
20    else ...                              ---- if necessary, the scheduler is updated
21    fi .
```

Figure 11: Task initiation rule.

by this task are available, which is checked with the allResourcesAvailable function (line 8), then a new token is generated with the task identifier and the task duration (line 12). Otherwise, the shifting mechanism is invoked (line 20)—note the ellipsis. If available, all required resources are removed from the resource set (grabResources function, line 18). Note also that rules update the information on execution time, task duration, etc. (see, e.g., the update of the task-tstamps attribute, lines 13–16).

A second rule, which models task completion, is triggered when there is a token for that task with zero time. In that case, the token is consumed and a new one is generated for the outgoing flow. All resources are released and all the message events associated to that task, if any, are added to the set of events.

Gateways. Split gateways are triggered when a token arrives in its incoming flow. Depending on the type of gate, tokens are placed in one of the outgoing flows, in several of them, or on all of them. The decision of which flows to activate is taken at random. For event-based gates, when a merge gateway is triggered, the incoming tokens are removed, a new token is added to the scheduler for the outgoing flow and simulation information is updated with synchronization times. A detailed description of the different rules handling the different types of gateways can be found, e.g., in [15].

Supervisor. The Simulation object is in charge of collecting the data on the chosen metric for the specified window of time (history length). A supervisor then analyzes the collected information and, if necessary, decides to update (increase or decrease) the number of resource instances. More details on the supervisor class and its resource-handling strategies are discussed in Section 5.3.

Workloads. Simulation-based analysis techniques are typically parameterized by the workload, which defines the rate at which new instances of a given process are executed. In a closed workload, a fixed number of tokens are injected in the process, corresponding to the number of times the process is to be executed.

Socket-based communication. A class CtrlSocket is in charge of the interaction with the predictor component. Every time a process starts or terminates, or the execution of a task begins or terminates, an event is sent to the predictor. As it will be seen in Section 5.3, when a prediction is due, the execution of the system stops and a special event is sent to the predictor component to notify that a prediction is due. See Section 5.2 for additional details.

5.2. Event-guided processes

The specification of processes presented in Section 5.1.2 includes elements that are not needed when the system is guided by a provided event log, such as probabilities, durations, and delays. However, the fact that both stages of the simulations—the autonomous execution described in Section 5.1 and the one described in this section—use the same representation greatly simplifies the specification. Thanks to this situation and even when the control of concurrency is different, the process is copied to evolve. In fact, the main difference between these two processes is that whilst in an autonomous process the execution is guided by tokens, inserted in the process by a workload manager object, in an event-guided process it is the events that guide the execution.

As pointed out in Section 4.1.1, event logs are sequences of event descriptions. Each event description includes a comma-separated sequence of values: a process identifier (a number), an event description, a timestamp, and information about each of the resources of the process. For each event, its name, number of available instances, queue

```
crl [initTask] :
   < PId : Process | nodes: (task(NId, TaskName, FId1, FId2, SE, RIds, SEI), Nodes), Atts >
   < SId : Simulation | task-tstamps: ..., gtime: T, resources: Rs, Atts1 >
   < LId : Ctrl | events: (event(TId, NId, event("init"), T') EL), Atts2 >
=> < PId : Process | nodes: (task(NId, TaskName, FId1, FId2, SE, RIds, SEI), Nodes), Atts >
   < SId : Simulation |
       task-tstamps: ...
       gtime: T',
       resources: grabResources(TId, NId, RIds, Rs, T'),
       Atts1 >
   < LId : Ctrl | events: EL, Atts2 > .
```

Figure 12: Rule initTask for processes guided by events.

size, and usage percentage is included. These event sequences are received through a socket, parsed, and then represented using appropriate declarations. For example, the event[7]

```
298, n004-init, 3703784394059892335/9007199254740992
```

is represented as

```
event("298", "n004", event("init"), 3703784394059892335/9007199254740992)
```

An object of class Ctrl keeps the list of events since it is in charge of the interaction with the predictor, reading the sequence of events throw sockets, and then guiding the execution in accordance with such events.

```
class Ctrl | events: List{LogEvent}, socket: Maybe{Oid}, buffer: String .
```

Once available in its events attribute, events will guide the execution by activating rules specifying the different actions that may occur in the system. For example, rule initTask in Figure 12 specifies the initiation of a task when an init task event is at the front of the event sequence. Although not shown in the rule to simplify the presentation, all the information on the execution (e.g., time stamps or resources) is gathered as in the autonomous simulations presented in Section 5.1. This information will be used, once the execution consumes all the events in the prediction, to update the number of instances of the resources. Note that the initTask rule mirrors quite closely that for the autonomous execution.

5.3. Resource adaptation based on LSTM predictions

F. Durán et al. [17] proposed a general scheme able to define adaptation strategies for resource assignment. They present different strategies, which are guided by,

[7] Real numbers are converted in the Python script to their representation as rationals to be handled by the Maude specification.

respectively, recent resource usage, recent resource requests, predicted behavior, and a combination of all available strategies. A thorough analysis on the different strategies is presented by them; in particular, the relevance of different parameters, such as the time between checks, the history length, the resource ranges, or adaptation thresholds are analyzed. In the strategy based on predictions proposed in [17], predictions are carried out by using the execution of the autonomous process itself, as presented in Section 5.1, by looking ahead before making a decision. The proposed approach follows the same general scheme of [17], but with predictions being provided by a properly trained neural network.

The above-mentioned scheme assumes that resource instances are taken from a pool when required if available. However, instead of assuming a fixed number of instances, new instances may be allocated or released to adjust the offer of resources to their demand, thus enabling the goal of minimizing costs. In this context, the amount of resources is periodically evaluated by looking at different metrics on the recent history or the current state of execution. In the strategies based on usage percentage or queue sizes in [17], the average of these values during the recent history of the process is taken into account. The fact that the allocation or deallocation of resources may take some time, looking ahead to anticipate the needs of the coming events may improve the average execution time and total cost of the process execution.

To specify such a mechanism, the Supervisor class provides the attributes time-between-checks and time-to-next-check to keep a timer, and check-interval to specify the length of the history to look at.

```
class Supervisor | time-between-checks: Time,
                   time-to-next-check: Time,
                   check-interval: Time .
```

The scheme assumes that decisions are taken in accordance to some given thresholds, which are also provided as parameters. The algorithm periodically checks if the value of the considered property is greater than the upper-bound threshold, in which case a new instance of the resource is allocated to the set of available resources; if it is smaller than the lower bound, then an instance is removed so that it is no longer available for use.

The Supervisor class is extended by subclass SupervisorPrediction to handle the new strategy. In addition to the thresholds attribute, with ranges for each resource type, it adds attributes look-ahead-time to consider different prediction sizes and forked-state to create an event-guided process to be executed on the prediction to be received from the Python predictor component.

```
class SupervisorPrediction |
         thresholds: Map{Id, Tuple{Float, Float}}, ---- usage thresholds
         look-ahead-time: Time
         forked-state: Maybe{System} .
subclass SupervisorPrediction < Supervisor .
```

The rules specifying the behavior of the supervisor object are shown in Figure 13. The supervisor-initiate-prediction rule is fired when the value of the time-to-next-check attribute is zero. It creates a copy of the part of the state needed for the event-guided execution (see Section 5.2): the Simulation object collects information on the execution, including timestamps and measure of resource usage, and the Process object. A new object of class CtrlSocket is created to read from the socket and collect the events to guide the execution. On the right-hand side of the rule there is a send message: a "PREDICT" event notifies the predictor that it is time to use the trace submitted until that time to feed the neural network, generate the prediction, and submit it through the socket.

To signal the end of the prediction, the predictor component sends an END event. When the CtrlSocket object in the forked-state attribute finds the END event, the second rule, supervisor-prediction-completed, is fired. It terminates the event-guided system and updates the resources by executing the update function. Basically, this function analyzes the resources along the execution of the prediction and decides whether it is a good idea to change the number of instances of each resource or not by referring to the thresholds provided. Finally, notice that the tokens are restored in the Simulation object so that the simulation can be resumed. The time-to-next-check timer is reset with the value of the time-between-checks attribute.

```
crl [supervisor-initiate-prediction] :
   < SId : Simulation | tokens: Tks, Atts1 >
   < PId : Process | Atts2 >
   < Sup : SupervisorPrediction |
      time-to-next-check: 0,           ---- check is due
      forked-state: null,
      look-ahead-time: T,
      Atts5 >
   < SH : SocketHandler | socket: SOCKET, Atts6 >
=> < SId : Simulation | Atts1 >        ---- tokens are removed to stop the simulation
   < PId : Process | Atts2 >
   < Sup : SupervisorPrediction |
      time-to-next-check: 0,
      forked-state:                    ---- a copy of the state is used to predict
        { < SId : Simulation | Atts1 > ---- no tokens, so regular rules cannot apply
          < PId : Process | Atts2 >
          < cs : CtrlSocket | socket: SOCKET, buffer: "", events: nil >
          Receive(SOCKET, cs) },
      look-ahead-time: T,
      tokens: Tks,                     ---- save tokens to restore the simulation
      Atts5 >
   < SH : SocketHandler | socket: SOCKET, Atts6 >
   send(SOCKET, SH, "PREDICT " + string(PREDICTION-TIME, 10) + "\n"
if Tks =/= nil .

rl [supervisor-prediction-completed] :
   < SId : Simulation | resources: Rs, gtime: T, Atts1 >
   < Sup : SupervisorPrediction |
      time-between-checks: TBC,
      time-to-next-check: 0,
      check-interval: CI,
      thresholds: Thds,
      forked-state:
        { < SId : Simulation | resources: Rs', gtime: T', Atts3 >
          < cs : CtrlSocket | buffer: "", events: END, Atts4 >
          Conf },
      tokens: Tks,                     ---- frozen tokens
      Atts2 >
=> < SId : Simulation |
      resources: update(Rs, Rs', Thds, TBC, CI, T, T'),
      gtime: T,
      tokens: Tks,                ---- tokens are restored to resume the simulation
      Atts1 >
   < Sup : SupervisorPrediction |
      time-between-checks: TBC,
      time-to-next-check: TBC,
      check-interval: CI,
      thresholds: Thds,
      forked-state: null,
      Atts2 > .
```

Figure 13: Predictive supervisor's rules.

6. Experimental Evaluation

This section compares the results of the LSTM-based predictive strategy with the strategies presented in [17] on three different cases. The results for the running example, the visa process, are presented in Section 6.1. Sections 6.2 and 6.3 present and discuss, respectively, a delivery system and a recruitment procedure. From the viewpoint of the analysis conducted on resource allocation strategies, the main differences between these processes are in the resources they use and their parameters. The visa process uses only two resources, with very different ATs (10 and 1), and a limited number of instances (10 for both). The delivery process uses 5 resources, with different but rather similar ATs (5, 4, 3, 2, and 3), and no limit to the number of instances of any of the resource types. Finally, the recruitment process uses three resources, with different ATs (10, 2, and 5), and different thresholds to the number of resources (50, 20, and 2). They also present different distributions in the durations of their tasks, which leads to very different behaviours, and different levels of difficulty in the predictions. Section 6.4 closes the experimental section with a wrapping-up discussion about the results obtained for the different experiments.

As explained in previous sections, instead of controlling a real system, the resource-provisioning strategies are applied on a Maude process that simulates such systems. The provisioning strategies are executed on the autonomous system presented in Section 5.1. To make a fair comparison, the training of the LSTM neural network is also done using traces generated by the same process using the same parameters. Specifically, more than one hundred thousand traces were generated using the autonomous system and these were used to train the neural network. Since the prediction strategy in [17] uses the autonomous process itself, it can be considered as predicting its own behavior, which in practice may imply an advantage. All strategies were then used in the same conditions to be able to gather information on their effectiveness and compare them fairly.

In the following experiments, the strategy based on machine-learning prediction is tagged as predictive-ml-usage. In addition to the predictive-ml-usage strategy, the following strategies are considered: the predictive-usage strategy is the strategy that adapts the

allocation of resources taking into account the prediction provided by the autonomous process; the queues strategy takes into account the sizes of the queues of each of the resources; the usage strategy takes into account the percentage of use of the resources; and the usage-queues strategy takes into account both usages and queues. As pointed out in Section 5.1, the queues, usage, and usage-queues strategies consider, respectively, the average value of either queue size, usage, or both for the CI interval previous to the evaluation time. The predictive strategies predictive-ml-usage and predictive-usage take the size of the prediction as a parameter in the form of the amount of time to look ahead, LAT. The experiments are presented in the following sections.

6.1. Evaluation experiments for the visa case study

The traces for the training of the network are obtained by executing 1 024 simulations. Each of these simulations runs a population of 100 instances, initiated using a closed workload with an inter-arrival time following an exponential distribution with lambda 0.5. This makes a total of 102 400 process execution instances. These traces are processed and the event attributes are tabulated, as explained in Section 4.1. In this case, the number of attributes is 10: timestamp of the events, activity names, execution instance identifier, population (i.e., the number of individuals), and for each resource the type of resource, number of available instances, usage percentage, and queue size. The LSTM neural network was trained using this data as explained in Section 4 and the resulting model used for the prediction using the predictive-ml-usage strategy presented in Section 5.3.

Figure 14 presents the objective functions obtained by using each of the different strategies, using different values for the TBC (time between checks) and CI (interval of time to consider), or LAT (look-ahead time). The figure shows 100 different results for each strategy, considering all combinations of integer values of TBC in the range $[1, 10]$, and CI and LAT in the range $[1, 10]$. For example, the case TBC=2 CI/LAT=3 for the usage strategy shows the objective function for that strategy with TBC 2 and CI 3. The case TBC=3 CI/LAT=4 for the predictive-usage strategy shows the objective function for that strategy with TBC 3 and LAT 4. As for the rest of the parameters, all experiments shown were run with allocation times 10 and 1 for the employee and

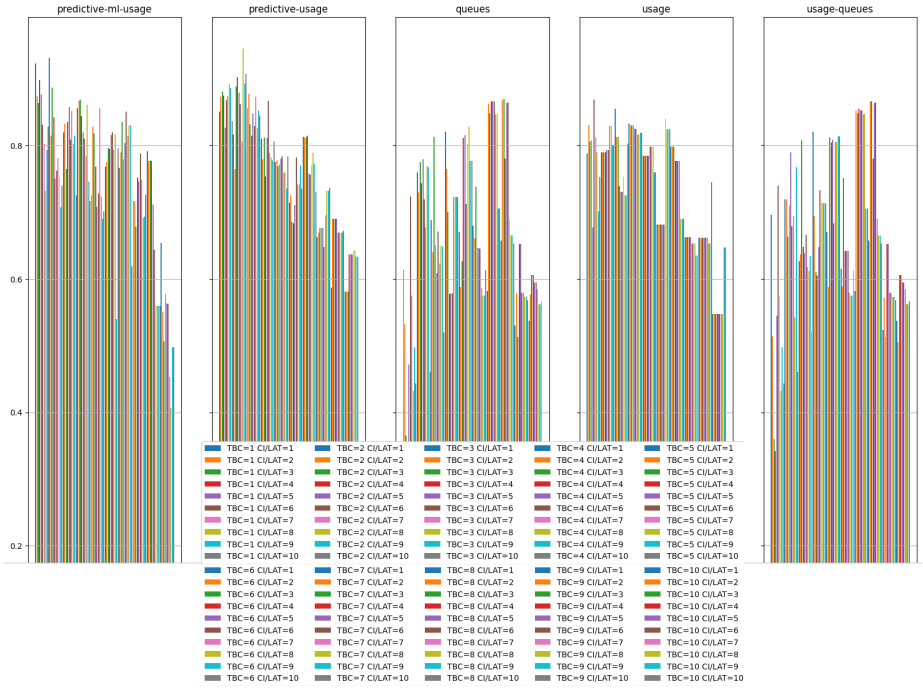

Figure 14: Objective function for the visa process with different parameters

the printer resources, respectively, ranges $[1, 10]$ for both, and costs 90 and 2 euros per hour, respectively. The thresholds considered were 70% and 80% for usage for both resources, and $[3, 6]$ for employees and $[2, 8]$ for printers. The objective function of each experiment in Figure 14 is calculated by normalizing the cost and execution time using a min-max normalization. The objective function is then calculated using the same weight 0.5 for both cost and execution time.

The specific value for each combination in Figure 14 is hard to see; the focus is on some specific cases below. However, the general pattern is quite informative, and facilitates some observations. All the strategies get quite good results, although both predictive strategies perform slightly better. There are some patterns that can be observed. For example, the predictions of the predictive-ml-usage strategy get worst as the TBC increases—the TBC of the experiments increases to the right, and for a same TBC, the CI/LAT also increases to the right. This is the case for all the other strategies, but the one based on queues, which does not behave well for small values of TBC.

	TBC	CI	Resources		Exec. time (h)		Cost (€)	Objective (Cost/Time)			
		LAT	Name	Usage	Avg.	Var.		50/50	60/40	40/60	Best
predictive-ml-usage	2	1	employee printer	68.17 36.24	24.29	0.38	106 299.24	0.93	0.92	0.94	1/1 0/100 1.0
	1	1	employee printer	68.98 35.01	23.97	0.34	110 685.77	0.92	0.91	0.94	
	1	4	employee printer	68.46 32.98	24.62	0.42	114 360.0	0.9	0.88	0.91	
predictive-usage	2	8	employee printer	73.86 40.43	25.07	1.46	97 267.04	**0.95**	**0.94**	**0.95**	2/8 0/100 0.96
	2	10	employee printer	73.41 36.95	25.67	1.6	105 380.57	0.91	0.9	0.91	
	2	4	employee printer	74.01 34.07	25.67	1.6	106 926.29	0.9	0.9	0.91	
queues	8	2	employee printer	77.45 39.45	30.02	2.89	91 333.04	0.87	0.89	0.85	7/2 99/1 1.0
	8	3	employee printer	77.53 39.45	30.02	2.89	90 680.22	0.87	0.89	0.85	
	7	4	employee printer	72.69 43.85	28.94	2.54	98 313.95	0.87	0.88	0.85	
usage	1	6	employee printer	66.97 27.3	24.88	1.41	121 840.37	0.87	0.85	0.89	1/6 0/100 0.96
	3	1	employee printer	66.77 37.54	26.14	1.73	118 577.04	0.86	0.84	0.87	
	6	7	employee printer	69.07 29.6	28.52	2.41	108 729.36	0.84	0.84	0.84	
usage-queues	8	2	employee printer	75.49 31.58	29.8	2.82	93 175.48	0.87	0.89	0.85	7/2 99/1 0.98
	8	3	employee printer	75.49 31.58	29.8	2.82	93 175.48	0.87	0.89	0.85	
	8	5	employee printer	72.03 41.77	29.03	2.57	98 393.31	0.86	0.88	0.85	

Table 1: Outputs of the best results for the visa example.

Table 1 presents data of a selection with the most relevant experiments. It shows the details for the cases in which each strategy got its best results in relation to the previous exhaustive experimentation. Since the results depend significantly of the weights given to the costs and the execution time when calculating the objective function, the experiments were carried out for the combinations of weights 0.0/1.0, 0.1/0.9, 0.2/0.8, ..., 1.0/0.0. The last four columns show the best values obtained for each strategy for the combinations 0.5/0.5, 0.6/0.4, 0.4/0.6 and the best one for each strategy for any weight combination. For instance, the first row corresponds to the experiment in which the predictive-ml-usage strategy is used with TBC 2 and LAT 1, which gave the best value for the objective function with weights 0.5/0.5. The observed usage percentages for the resources were 68.17% for employee and 36.24% for printer. The average execution time was 24.29 and the variance of the execution times was 0.38. The total cost of the execution was 106 299.24. Then, with weights 0.5/0.5 the objective function was 0.93, with weights 0.6/0.4 the objective function was 0.92, and with weights 0.4/0.6 the objective function was 0.94. Still for the predictive-ml-usage strategy, the second row shows the best value for the objective function with weights 0.6/0.4, and the third one for weights 0.4/0.6. The last row corresponds to the best result obtained for the corresponding strategy. For these, no details on the result are shown, only the TBC/CI/LAT value, the weights, and the objective function. For example, for the predictive-ml-usage strategy, the best value of the objective function was 1, obtained with TBC 1, LAT 1, weights 0.0/1.0. This means that the best time was obtained and in isolation, this strategy got the best value. The best results are obtained by the predictive-usage strategy, although followed very closely by the predictive-ml-usage strategy. The queues strategy seems to make a better use of the resources, possibly providing better stability on their use. With greater TBC values (7 and 8), and CI between 2 and 4, it obtains the best results. However, as pointed out before, it performs a bit worst than the prediction-based strategies. The best value obtained by the queues strategy is also 1, although with weights 0.99/0.01, which indicates that is the best strategy for minimizing costs.

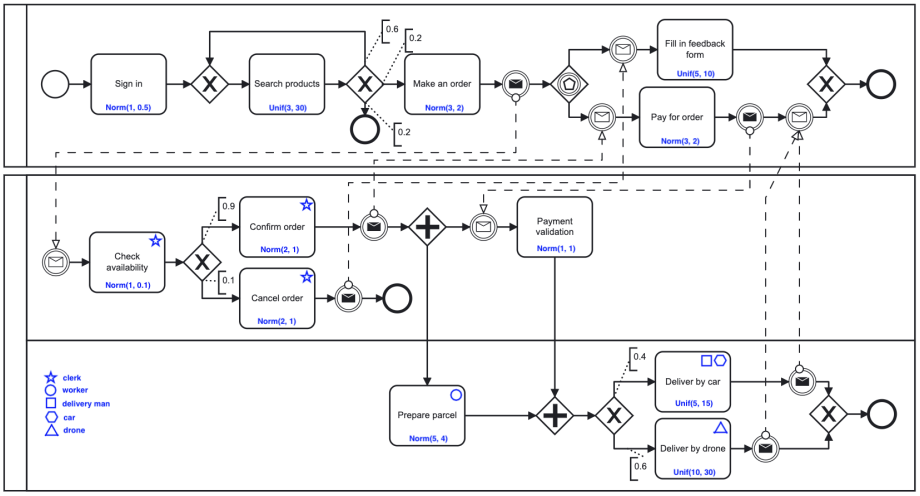

Figure 15: Delivery process.

6.2. An online store: the delivery process

Figure 15 presents a BPMN process modeling an online store. In this process, the client first signs in and then repeatedly looks for products. Eventually, the client can decide to give up or to make an order by submitting it to the order management lane. The client then waits for a response (i.e., acceptance or refusal of this order). If the order can be completed, then the parcel is received and the client pays for it. Otherwise (i.e., the order is refused), the client fills in a feedback form. As far as the management lane is concerned, the first task aims at verifying whether the goods ordered by the client are available. If they are not available, then the order is canceled; otherwise, the order is confirmed. The order management takes care of the payment of the order, whereas the delivery lane is triggered to prepare the parcel to be delivered. The delivery may be carried out by car or by drone.

In this case, five different resources are used by the different tasks, namely clerks, workers, delivery men, cars, and drones. In this case, no limit is given to the allocation of resources; instances can increase and decrease as necessary. Regarding allocation time, it is 5 for clerks, 4 for workers, 3 for couriers, 2 for cars, and 3 for drones. The icon on the right-top corner of each task indicates the resources necessary for its

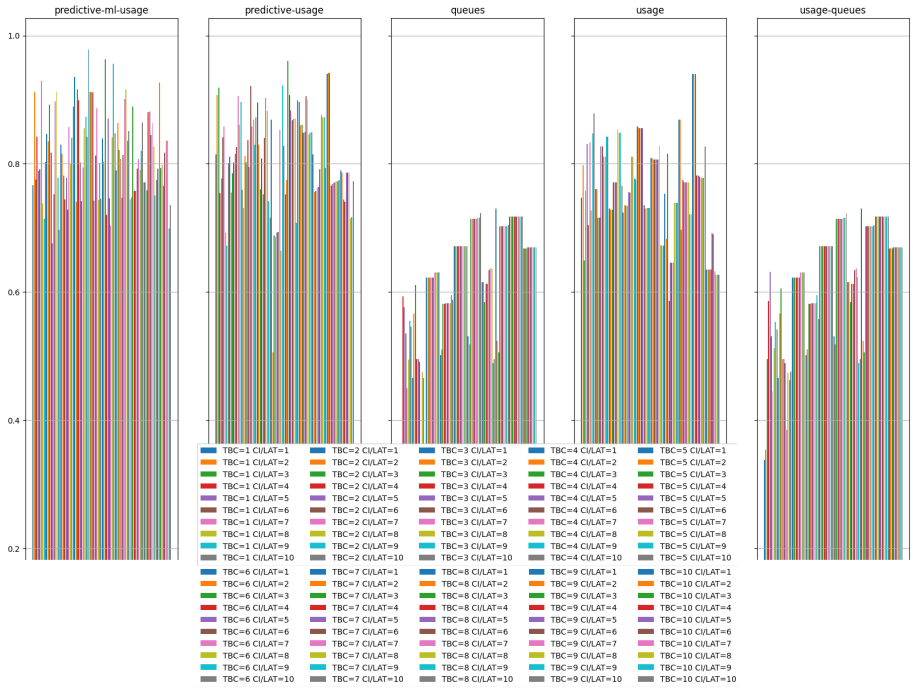

Figure 16: Objective function for the delivery process

execution. As for the previous example, branches of exclusive gates have a probability value, and tasks' duration is specified as probability distributions shown at the bottom of each task. Also as for the visa process, all flows have a delay given by a normal distribution with average 1 and standard deviation 0.5.

A process similar to the one followed for the visa example is followed for the delivery case. After training the LSTM neural network with traces generated using the autonomous process, the model is used to execute the process using the predictive-ml-usage strategy with the other four strategies using similar parameters and assumptions. Figure 16 is to the delivery example what Figure 14 is to the visa case. Again, in this case, the prediction-based strategies seem to behave better than the others.

Table 2 focus on the experiments with best results for the delivery case, exactly as done with the visa case in Table 1. The predictive-ml-usage strategy outperforms all the others, giving the best values for the three combinations of weights. There are five resource types, with no limit to the number of instances, and allocation times quite

similar, in the range 1-5.

6.3. A recruitment process

The BPMN process depicted in Figure 17 describes the recruitment process in an enterprise. This process shows how a candidate must fill in a hiring form, carries on a medical checkup, and, in some cases, applies for a visa. Once the documentation is submitted, it is checked by the human resources office, which can decide to accept, reject, or request additional documentation. If accepted, the candidate is informed, an assistant is in charge of preparing a welcome kit, and the technical staff is in charge of including the corresponding data in the enterprise's DB. As for the other processes, the graphical representation of the process includes the resources used and the durations of the tasks. In this example, there are three resources; namely, human resources, assistants, and technical staff. In this case, the number of instances is limited to 50, 20, and 2 for hr, assistant, and technicalstaff, respectively. Allocation times are set to 10, 2, and 5 for hr, assistant, and technicalstaff, respectively. Also as for the other sample processes in this work, the delays of all flows is modelled by a distribution Norm(1, 0.5) (although this information is not shown in the picture to improve readability).

The five strategies are used with the same parameters and assumptions. Figure 18 is to the recruitment example what Figures 14 and 16 were to the corresponding cases. Again in this case, the prediction-based strategies seem to behave better than the others, although the usage-based strategy shows a very similar behavior. It can be observed that the queues-based strategies show more unstable results as TBC and CI increase.

Table 3 shows the details of the experiments for the recruitment case, as done with the other examples. In this case, even though Figure 18 shows a quick degradation of the results for the queues strategy as the TBC grows, this strategy outperforms the others when the focus is on the best combinations, giving the best values for the three combinations of weights highlighted. As seen in the two previous examples, the queues strategy is quite good at minimizing costs; in this case, costs are much higher than in other examples and also have a greater range. However, the variance is much lower. Indeed, for the case TBC=1/CI=1, the variance of the experiments was 0. Even though the predictive strategies produce in general better execution times, the reduction in cost

	TBC	CI	Resources		Exec. time (h)		Cost (€)	Objective (Cost/Time)			
		LAT	Name	Usage	Avg.	Var.		50/50	60/40	40/60	Best
predictive-ml-usage	5	1	car	35.2	62.92	16.15	91160.99	**0.98**	**0.98**	**0.98**	4/1 0/100 1.0
			clerk	34.99							
			courier	35.69							
			drone	51.08							
			worker	38.25							
	6	3	car	34.99	63.07	16.27	95 062.71	0.96	0.96	0.97	
			clerk	38.75							
			courier	36.03							
			drone	49.76							
			worker	34.86							
predictive-usage	6	3	car	32.41	64.06	17.08	91 629.26	0.96	0.96	0.96	4/1 100/0 0.98
			clerk	35.17							
			courier	33.23							
			drone	49.94							
			worker	37.43							
	9	3	car	34.3	64.47	17.42	95 440.6	0.94	0.94	0.94	
			clerk	35.94							
			courier	34.63							
			drone	42.03							
			worker	45.99							
queues	8	1	car	33.7	77.43	29.91	102 738.86	0.73	0.76	0.7	4/9 99/1 0.95
			clerk	36.84							
			courier	33.7							
			drone	63.17							
			worker	68.31							
	9	9	car	35.47	78.48	31.07	101 872.14	0.72	0.75	0.68	
			clerk	37.5							
			courier	36.53							
			drone	48.2							
			worker	55.03							
usage	9	3	car	34.3	64.47	17.42	95 887.46	0.94	0.94	0.94	9/1 0/100 0.95
			clerk	35.94							
			courier	34.64							
			drone	42.04							
			worker	45.99							
	9	2	car	34.3	64.47	17.42	95 887.46	0.94	0.94	0.94	
			clerk	35.94							
			courier	34.64							
			drone	42.04							
			worker	45.99							
usage-queues	8	1	car	33.7	77.43	29.91	102 738.86	0.73	0.76	0.7	4/9 99/1 0.95
			clerk	36.84							
			courier	33.7							
			drone	63.17							
			worker	68.31							
	9	9	car	35.47	78.48	31.07	101 872.14	0.72	0.75	0.68	
			clerk	37.5							
			courier	36.53							
			drone	48.2							
			worker	55.03							

Table 2: Output of the best results for the delivery process.

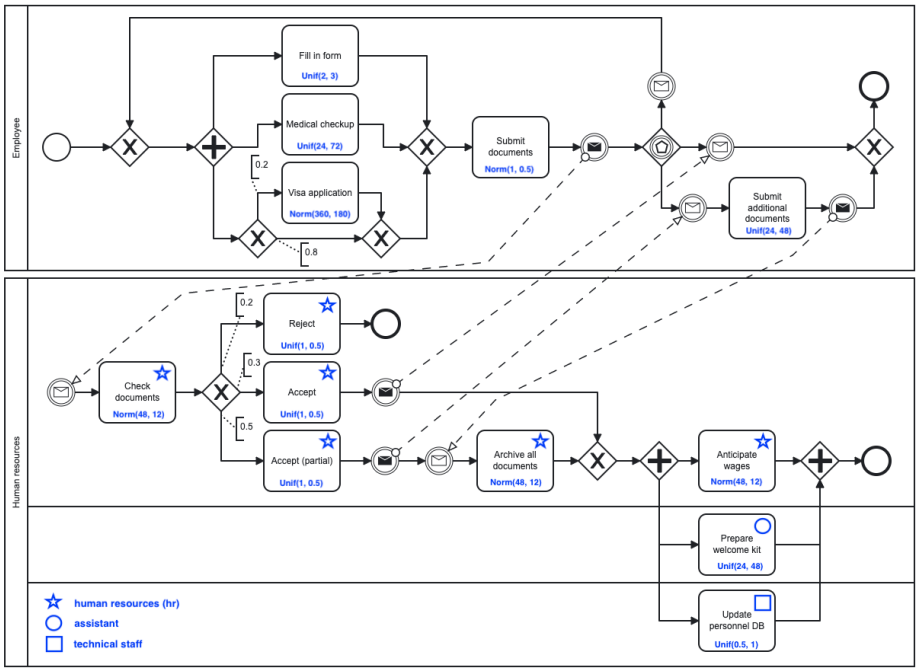

Figure 17: Recruitment process.

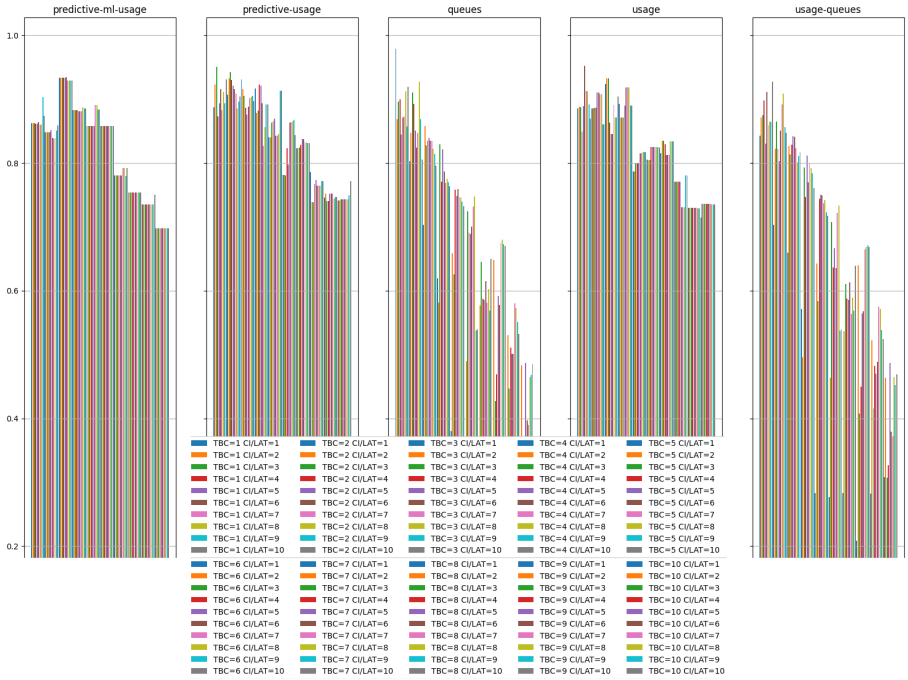

Figure 18: Objective function for Recruitment process

makes a difference in this case. Note that the numbers for the predictive strategies are very close.

6.4. Discussion on the experiments

Section 4.2 explains how the best possible structure and values were found for the LSTM model for the visa process using the Hyperband algorithm. A similar strategy has been followed for the delivery and recruitment processes: the best sequence lengths for these processes are 6 and 18, respectively. The results of the training can be seen in Figure 19. In this figure, it can be observed how the learning for the visa and delivery processes were quite good. There is a greater margin of error for the recruitment process, due to the difficulty of predicting the duration of the activities.

The experimental evaluation shows that the different strategies work reasonably well. Especially, the one based on machine-learning predictions present a consistently exceptional behavior, with values of over 0.9 for the objective function with appropriate parameters. The training of the models is of course key to having this good behavior.

	TBC	CI LAT	Resources Name	Resources Usage	Exec. time (h) Avg.	Exec. time (h) Var.	Cost (€)	Objective (Cost/Time) 50/50	60/40	40/60	Best
predictive-ml-usage	3	8	assistant hr technicalstaff	41.82 66.14 6.43	302.73	236.04	1 878 961.80	0.93	0.92	0.93	
	3	2	assistant hr technicalstaff	38.49 66.15 6.01	300.84	230.28	1 862 235.37	0.93	0.93	0.94	3/1 0/100 0.96
	3	9	assistant hr technicalstaff	41.81 66.21 6.43	302.73	236.04	1 876 664.68	0.93	0.92	0.93	
predictive-usage	1	3	assistant hr technicalstaff	35.63 53.4 5.3	280.48	2.16	1 880 548.27	0.95	0.94	0.96	
	2	3	assistant hr technicalstaff	52.52 72.35 8.08	288.88	183.88	1 879 851.87	0.94	0.93	0.95	1/3 0/100 1.0
	1	10	assistant hr technicalstaff	41.33 72.66 6.47	302.08	53.17	1 867 500.20	0.93	0.93	0.94	
queues	1	1	assistant hr technicalstaff	53.37 90.11 6.87	301.62	0.0	1 590 624.98	**0.98**	**0.98**	**0.97**	
	2	8	assistant hr technicalstaff	54.73 81.08 6.27	318.39	0.33	1 797 226.10	0.93	0.93	0.93	1/1 89/11 1.0
	1	10	assistant hr technicalstaff	63.12 74.28 6.51	313.04	5.18	1 873 075.87	0.92	0.92	0.92	
usage	1	6	assistant hr technicalstaff	47.46 66.98 8.44	281.87	8.93	1 862 707.55	0.95	0.94	0.96	
	3	3	assistant hr technicalstaff	41.13 60.08 6.17	291.94	23.23	1 917 040.75	0.93	0.92	0.94	1/6 0/100 1.0
	3	2	assistant hr technicalstaff	41.13 60.08 6.17	291.94	23.23	1 917 040.75	0.93	0.92	0.94	
usage-queues	1	10	assistant hr technicalstaff	65.92 74.51 8.7	300.16	0.98	1 904 629.59	0.93	0.92	0.93	
	1	6	assistant hr technicalstaff	67.01 82.36 6.77	339.94	45.69	1 764 833.90	0.91	0.92	0.91	1/10 0/100 0.96
	2	8	assistant hr technicalstaff	48.01 82.75 6.23	330.54	3.19	1 833 753.85	0.91	0.91	0.91	

Table 3: Outputs of the best results for the recruitment example.

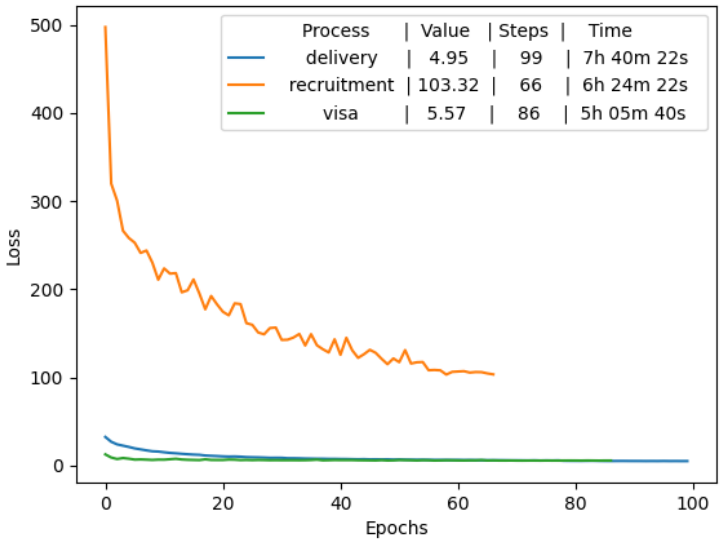

Figure 19: Loss for the train process.

Indeed, the fact that the loss is greater in the recruitment case may be part of the cause why it is surpassed by the other strategies. However, it only explains the results in part. It may be concluded that some reasons lay in other places rather than solely on the strategies themselves. The number of resources, the restrictions on those resources (ranges and allocation times), and the structure of the process can be key for the results obtained by the different strategies. Even though the prediction of activities is almost perfect, the prediction of durations and delays becomes more difficult when more alternative behaviors are possible due to the high concurrency of the process instances, and also when the process execution gets saturated. In general, however, the predictive methods behave better and are more stable, even more when resource allocation times are larger. It seems that predicting the need and deciding on resource allocations ahead of time is key to avoid many and lengthy delays.

One main conclusion of the experimental evaluation is that parameters like the TBC or CI/LAT are key for optimal results. Depending on the parameters of the problem, the values identifying better results will be different. Thus, for example, for the visa

case, the predictive-ml-usage strategy shows the best results for a TBC of 1-2, with a small value of LAT. The queues strategy, however, seems to work better with greater TBC values, specifically for 7-8. For the delivery case, the situation was different. In this case, the predictive-ml-usage strategy shows the best results for a TBC of 5-6. The other strategies also work better with bigger TBC values. In the recruitment case the situation is the opposite. In this case the predictive-ml-usage strategy shows the best results for a TBC of 3, whilst the others work better with smaller values for TBC. In this case, all strategies behave properly with large values of CI/LAT.

7. Concluding Remarks

The results presented in this paper are part of a long-standing effort to provide BPMN modeling with extensions and formal analysis tools based on rewriting logic [30]. This paper, as an extended version of [10], explores the use of deep learning as a complement to the existing wealth of prediction heuristic for BPMN resource allocation and occupancy. It combines rewriting logic with *predictive business process monitoring*, a family of (online) process monitoring techniques that seek to predict the future state or properties of ongoing cases of a process based on models extracted from historical cases recorded in event logs [44].

The approach proposed in this paper is focused on next-step and trace prediction to anticipate the availability and need of resource replicas. Long short-term memory neural networks (LSTM) [46] are used for trace prediction. The idea is that, given a concurrent process configuration, trained LSTMs are used as black boxes to query for potential next steps of process instance to assess resource replica availability within a given time window, in combination with a formal rewriting logic semantics of BPMN executable in the Maude language/system. It is shown that the proposed LSTM-based heuristic is competitive and, in some cases, produces better results than some of the above-mentioned bounded state-space search and waiting queues based prediction heuristics.

The extensive experimentation suggests that even though the new strategy is not always better than other alternatives, it is remarkably stable in performance, and is very close to the best one if it is not the best one itself. A proper LSTM training is key

for optimal results, and a good balance between the amount of data and its quality was found for feeding the neural network training. The selection of parameters, which may require dynamic adjustment if the conditions change, is also key.

As for future work, the plan is to advance in several lines of research. The first line involves the development of new allocation strategies. Since forecasts get worst when looking further into the future, a combination of predictions heuristics is being explored to decide on resource allocations. Since neural networks provide a number of predictions, each with a probability, instead of taking the best one and taking it as the good one, a new strategy may consider several of these values and combine them in different ways. Further experimentation is of course required, not only with more and more complex processes, but also with real systems at real time. As explained in the introduction of this work, the system is developed to interact with real systems using real traces. To use real logs by real systems to allocate their resources in real time, three main problems are to be faced. First, deep learning requires huge amounts of curated training data, which is not easy to get. Second, a detailed stable workflow is assumed, which companies are more or less reluctant to provide. Then, there must be mechanisms to act on the system to handle their resources. These issues are interesting research problems worth being considered.

Another future research direction is the use of deep learning techniques for business process optimization in a sense different to the one explored here. F. Durán and G. Salaün have explored in [19] the structural optimization of BPMN processes using automated refactoring, guided by different strategies. The authors strongly believe that deep learning methods can be used also for structural optimization of processes under given and well crafted constraints.

Acknowledgments. The authors would like to thank the reviewers for carefully reading the manuscript; their comments have been of great help in improving its quality and clarity. The first two authors has been partially supported by projects TED2021-130666B-I00 and PID2021-125527NB-I00 funded by the Spanish government. The work of Rocha was partially funded by the Minciencias (Ministerio de Ciencia Tecnología e Innovación, Colombia) project PROMUEVA (BPIN 2021000100160).

References

[1] IEEE Std 1849™-2016 standard for eXtensible event stream (XES) for achieving interoperability in event logs and event streams, September 2016.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. `https://www.tensorflow.org/`.

[3] G. Agha, J. Meseguer, and K. Sen. PMaude: Rewrite-based Specification Language for Probabilistic Object Systems. *ENTCS*, 153(2):213 – 239, 2006.

[4] M. Camargo, M. Dumas, and O. González-Rojas. Learning accurate LSTM models of business processes. In T. Hildebrandt, B. F. van Dongen, M. Röglinger, and J. Mendling, editors, *Business Process Management*, pages 286–302. Springer, 2019.

[5] M. Camargo, M. Dumas, and O. González-Rojas. Learning accurate business process simulation models from event logs via automated process discovery and deep learning. In X. Franch, G. Poels, F. Gailly, and M. Snoeck, editors, *Advanced Information Systems Engineering*, pages 55–71. Springer, 2022.

[6] F. Chollet et al. Keras, 2015. `https://github.com/fchollet/keras`.

[7] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude*, volume 4350 of *LNCS*. Springer, 2007.

[8] M. de Leoni, W. M. P. van der Aalst, and B. F. van Dongen. Data- and resource-aware conformance checking of business processes. In W. Abramowicz, D. Kriksciuniene, and V. Sakalauskas, editors, *Business Information Systems*

- *15th International Conference, BIS. Proceedings*, volume 117 of *Lecture Notes in Business Information Processing*, pages 48–59. Springer, 2012.

[9] C. Di Francescomarino, C. Ghidini, F. M. Maggi, G. Petrucci, and A. Yeshchenko. An eye into the future: leveraging a-priori knowledge in predictive business process monitoring. In *International Conference on Business Process Management*, pages 252–268. Springer, 2017.

[10] F. Durán, D. Martínez, and C. Rocha. Business processes analysis with resource-aware machine learning scheduling in rewriting logic. In K. Bae, editor, *Rewriting Logic and Its Applications*, pages 113–129. Springer, 2022.

[11] F. Durán, C. Rocha, and G. Salaün. Computing the parallelism degree of timed BPMN processes. In *Software Technologies: Applications and Foundations - STAF 2018*, volume 11176 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2018.

[12] F. Durán, C. Rocha, and G. Salaün. Stochastic analysis of BPMN with time in rewriting logic. *Sci. Comput. Program.*, 168:1–17, 2018.

[13] F. Durán, C. Rocha, and G. Salaün. Symbolic specification and verification of data-aware BPMN processes using rewriting modulo SMT. In *12th International Workshop on Rewriting Logic and Its Applications - WRLA 2018*, volume 11152 of *Lecture Notes in Computer Science*, pages 76–97. Springer, 2018.

[14] F. Durán, C. Rocha, and G. Salaün. Analysis of resource allocation of BPMN processes. In *17th International Conference on Service-Oriented Computing - ICSOC 2019*, volume 11895 of *Lecture Notes in Computer Science*, pages 452–457. Springer, 2019.

[15] F. Durán, C. Rocha, and G. Salaün. A rewriting logic approach to resource allocation analysis in business process models. *Sci. Comput. Program.*, 183, 2019.

[16] F. Durán, C. Rocha, and G. Salaün. Analysis of the runtime resource provisioning of BPMN processes using Maude. In *13th International Workshop on Rewriting*

Logic and Its Applications - WRLA 2020, volume 12328 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2020.

[17] F. Durán, C. Rocha, and G. Salaün. Resource provisioning strategies for BPMN processes: Specification and analysis using Maude. *J. Log. Algebraic Methods Program.*, 123:100711, 2021.

[18] F. Durán and G. Salaün. Verifying timed BPMN processes using Maude. In *Proc. of COORDINATION*, volume 10319 of *LNCS*, pages 219–236. Springer, 2017.

[19] F. Durán and G. Salaün. Optimization of BPMN processes via automated refactoring. In J. Troya, B. Medjahed, M. Piattini, L. Yao, P. Fernández, and A. Ruiz-Cortés, editors, *Service-Oriented Computing - 20th International Conference, ICSOC 2022, Seville, Spain, November 29 - December 2, 2022, Proceedings*, volume 13740 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2022.

[20] F. Durán, N. Pozas, and C. Rocha. BPMN resource manager. `https://github.com/Pozas91/bpmn-resource-manager`.

[21] C. D. Francescomarino, C. Ghidini, F. M. Maggi, and F. Milani. Predictive process monitoring methods: Which one suits me best? In M. Weske, M. Montali, I. Weber, and J. vom Brocke, editors, *Business Process Management - 16th International Conference, BPM, Proceedings*, volume 11080 of *Lecture Notes in Computer Science*, pages 462–479. Springer, 2018.

[22] M. Hinkka, T. Lehto, and K. Heljanko. Exploiting event log event attributes in RNN based prediction. In T. Welzer, J. Eder, V. Podgorelec, R. Wrembel, M. Ivanović, J. Gamper, M. Morzy, T. Tzouramanis, J. Darmont, and A. Kamišalić Latifić, editors, *New Trends in Databases and Information Systems*, pages 405–416. Springer, 2019.

[23] M. Hinkka, T. Lehto, K. Heljanko, and A. Jung. Classifying process instances using recurrent neural networks. In F. Daniel, Q. Z. Sheng, and H. Motahari, editors, *Business Process Management Workshops*, pages 313–324. Springer, 2019.

[24] Z. Huang, W. M. P. van der Aalst, X. Lu, and H. Duan. Reinforcement learning based resource allocation in business process management. *Data Knowl. Eng.*, 70(1):127–145, 2011.

[25] ISO/IEC. International Standard 19510, Information technology – Business Process Model and Notation. 2013.

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[27] W. Kratsch, J. Manderscheid, M. Röglinger, and J. Seyfried. Machine learning in business process monitoring: A comparison of deep learning and classical approaches used for outcome prediction. *Bus. Inf. Syst. Eng.*, 63(3):261–276, 2021.

[28] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[29] N. Mehdiyev, J. Evermann, and P. Fettke. A novel business process prediction model using a deep learning method. *Bus. Inf. Syst. Eng.*, 62(2):143–157, 2020.

[30] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[31] J. Nakatumba and W. M. P. van der Aalst. Analyzing resource behavior using process mining. In S. Rinderle-Ma, S. W. Sadiq, and F. Leymann, editors, *Business Process Management Workshops, BPM 2009 International Workshops, Revised Papers*, volume 43 of *Lecture Notes in Business Information Processing*, pages 69–80. Springer, 2009.

[32] OASIS. Web Services - Human Task (WS-HumanTask) Version 1.1. 2010.

[33] OASIS. WS-BPEL Extension for People (BPEL4People) Specification Version 1.1. 2010.

[34] V. Pasquadibisceglie, A. Appice, G. Castellano, and D. Malerba. Using convolutional neural networks for predictive process analytics. In *2019 International Conference on Process Mining (ICPM)*, pages 129–136, 2019.

[35] P. Pfeiffer, J. Lahann, and P. Fettke. Multivariate business process representation learning utilizing gramian angular fields and convolutional neural networks. In *19th International Conference on Business Process Management - BPM 2021*, volume 12875 of *Lecture Notes in Computer Science*, pages 327–344. Springer, 2021.

[36] A. Pika, M. T. Wynn, C. J. Fidge, A. H. M. ter Hofstede, M. Leyer, and W. M. P. van der Aalst. An extensible framework for analysing resource behaviour using event logs. In M. Jarke, J. Mylopoulos, C. Quix, C. Rolland, Y. Manolopoulos, H. Mouratidis, and J. Horkoff, editors, *Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Proceedings*, volume 8484 of *Lecture Notes in Computer Science*, pages 564–579. Springer, 2014.

[37] C. Rocha, J. Meseguer, and C. Muñoz. Rewriting Modulo SMT and Open System Analysis. *Journal of Logical and Algebraic Methods in Programming*, 86(1):269 – 297, 2017.

[38] N. Russell and W. M. P. van der Aalst. Work distribution and resource management in BPEL4People: Capabilities and opportunities. In Z. Bellahsene and M. Léonard, editors, *Advanced Information Systems Engineering, 20th International Conference, CAiSE, Proceedings*, volume 5074 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2008.

[39] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond. Workflow resource patterns: Identification, representation and tool support. In O. Pastor and J. F. e Cunha, editors, *Advanced Information Systems Engineering, 17th International Conference, CAiSE, Proceedings*, volume 3520 of *Lecture Notes in Computer Science*, pages 216–232. Springer, 2005.

[40] E. R. Taghiabadi, V. Gromov, D. Fahland, and W. M. P. van der Aalst. Compliance checking of data-aware and resource-aware compliance requirements.

In R. Meersman, H. Panetto, T. S. Dillon, M. Missikoff, L. Liu, O. Pastor, A. Cuzzocrea, and T. K. Sellis, editors, *On the Move to Meaningful Internet Systems: OTM Conferences - Confederated International Conferences: CoopIS, and ODBASE, Proceedings*, volume 8841 of *Lecture Notes in Computer Science*, pages 237–257. Springer, 2014.

[41] H. Tan and W. M. P. van der Aalst. Implementation of a YAWL work-list handler based on the resource patterns. In *10th International Conference on CSCW in Design (CSCWD)*, pages 1184–1189. IEEE, 2006.

[42] N. Tax, I. Teinemaa, and S. J. van Zelst. An interdisciplinary comparison of sequence modeling methods for next-element prediction. *Softw. Syst. Model.*, 19(6):1345–1365, 2020.

[43] I. Teinemaa, M. Dumas, M. L. Rosa, and F. M. Maggi. Outcome-oriented predictive process monitoring: Review and benchmark. *ACM Trans. Knowl. Discov. Data*, 13(2):17:1–17:57, 2019.

[44] I. Verenich, M. Dumas, M. L. Rosa, F. M. Maggi, and I. Teinemaa. Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. *ACM Trans. Intell. Syst. Technol.*, 10(4):34:1–34:34, 2019.

[45] C. Walck. Hand-Book on Statistical Distributions for Experimentalists. Technical Report SUF–PFY/96–01, Universitet Stockholms, Stockholm, Sept. 2007.

[46] P. Wilmott. *Machine Learning: An applied mathematics introduction*. Panda Ohana, 2019.

[47] J. Yang, C. Ouyang, A. H. M. ter Hofstede, and W. M. P. van der Aalst. No time to dice: Learning execution contexts from event logs for resource-oriented process mining. In C. D. Ciccio, R. M. Dijkman, A. del-Río-Ortega, and S. Rinderle-Ma, editors, *Business Process Management - 20th International Conference, BPM, Proceedings*, volume 13420 of *Lecture Notes in Computer Science*, pages 163–180. Springer, 2022.