Modeling Extraneous Activity Delays in Business Process Simulation

1st David Chapela-Campa
University of Tartu
Tartu, Estonia
david.chapela@ut.ee

2nd Marlon Dumas
University of Tartu
Tartu, Estonia
marlon.dumas@ut.ee

***Abstract*—Business Process Simulation (BPS) is a common approach to estimate the impact of changes to a business process on its performance measures. For example, BPS allows us to estimate what would be the cycle time of a process if we automated one of its activities. The starting point of BPS is a business process model annotated with simulation parameters (a BPS model). Several studies have proposed methods to automatically discover BPS models from event logs via process mining. However, current techniques in this space discover BPS models that only capture waiting times caused by resource contention or resource unavailability. Oftentimes, a considerable portion of the waiting time in a business process is caused by extraneous delays, e.g. a resource waits for the customer to return a phone call. This paper proposes a method that discovers extraneous delays from input data, and injects timer events into a BPS model to capture the discovered delays. An empirical evaluation involving synthetic and real-life logs shows that the approach produces BPS models that better reflect the temporal dynamics of the process, relative to BPS models that do not capture extraneous delays.**

***Index Terms*—Business process simulation, process mining**

I. Introduction

Business Process Simulation (BPS) is an analysis technique that enables users to address "what-if" analysis questions, such as "what would be the cycle time of the process if the rate of arrival of new cases doubles?" or "what if 10% of the workforce becomes unavailable for an extended time period?". The starting point of BPS is a process model (e.g. in the Business Process Model and Notation – BPMN)[1] enhanced with simulation parameters (a BPS model). Traditionally, BPS models are manually created by modeling experts. This task is time-consuming and error-prone [1]. To tackle this shortcoming, several studies have advocated for the use of process mining techniques to automatically discover BPS models from business process event logs [2], [3], [4].

Current approaches for automated BPS model discovery produce BPS models under the assumption that all waiting times are caused either by unavailability of resources (e.g. resources being off-duty) or by resource contention (all resources who can perform an activity instance are busy). In practice, there are other sources of waiting times in business processes. For example, Andrews et al. [5] found that a large proportion of waiting time (a.k.a. "shelf time") in real business processes is not caused by the availability or capacity of resources, but by other factors, such as waiting for a customer to respond. Moreover, it is often the case that these waiting times, herein called *extraneous delays*, are not explicitly recorded in the log.

In this setting, this paper studies the hypothesis that taking into account extraneous waiting times during automated BPS model discovery, leads to BPS models that better reflect the temporal performance of the process. Specifically, the paper addresses the following problem: given an event log L (wherein each activity instance has a resource, start time, and end time) and a BPS model M of a process, compute an enhanced BPS model M' by including waiting times caused by extraneous factors observed in the event log, such that M' replicates the behavior recorded in L more accurately than M. To address this problem, we propose a technique that performs this enhancement by adding, to the BPS model, a timer event before each activity that has delays that cannot be explained by resource contention. To model the extraneous delay of each activity, our proposal first estimates the instant in time in which its activity instances could have started (henceforth, its *optimal start*) using the control-flow and resource allocation information available in L. Then, we consider as extraneous waiting time the interval between the earliest start and the recorded start of each activity instance. Finally, based on these extraneous delay estimations, we fit a probability distribution function to capture the duration of the injected timer event.

The paper reports on an evaluation that assesses: *i)* the ability of the proposed technique to accurately discover extraneous delays known to be present in a log; and *ii)* the extent to which a BPS model that incorporates the extraneous delays discovered by the technique, more closely replicates the temporal dynamics in a log, relative to a baseline BPS model.

The rest of the paper is structured as follows. Sec. II introduces basic notions of process mining and business process simulation, and gives an overview of prior related research. Sec. III describes the proposed approach. Sec. IV discusses the empirical evaluation, and Sec. V draws conclusions and sketches future work.

II. Background and Related Work

This section introduces basic notions of process mining related to the problem that is being addressed, and describes existent techniques related to the enhancement of BPS models.

This research is funded by the European Research Council (PIX Project).

[1]https://www.bpmn.org/

2022 4th International Conference on Process Mining (ICPM 2022), ISBN 979-8-3503-9714-7

A. Event Log and Process Mining

We consider a business process that involves a set of *activities* A. We denote each of these activities with α. An *activity instance* $\varepsilon = (\varphi, \alpha, \tau_s, \tau_e, \rho)$ denotes an execution of the activity α, where φ identifies the *process trace* (i.e. the execution of the process) to which this event belongs to, τ_s and τ_e denote, respectively, the instants in time in which this activity instance started and ended, and ρ identifies the *resource* that performed the event. Accordingly, we write $\varphi(\varepsilon_i)$, $\alpha(\varepsilon_i)$, $\tau_s(\varepsilon_i)$, $\tau_e(\varepsilon_i)$, and $\rho(\varepsilon_i)$ to denote, respectively, the process trace, the activity, the start time, the end time, and the resource associated with the activity instance ε_i. We denote with $\omega(\varepsilon_i)$ the *waiting time* of ε_i, representing the time since ε_i became available for processing, until its recorded start ($\tau_s(\varepsilon_i)$). The *cycle time* of a trace (a.k.a. the trace duration) is the difference between the largest τ_e and the smallest τ_s of the activity instances observed in the trace. An *activity instance log* L is a collection of activity instances recording the information of the execution of a set of traces of a business process. Table I shown an example of 10 activity instances from an activity instance log.

It must be noted that, usually, the process execution information is recorded in event logs, containing the information of each activity instance as independent events, one representing its start, and another one representing its end. Furthermore, there are cases in which event logs also store events recording other stages of the activity instances' lifecycle, e.g. schedule. In this paper, we work with the concept of activity instance log to group the information of each activity instance in a single element. The transformation from a typical event log to an activity instance log is straightforward, and it has been already described in other research [6]. Nevertheless, we will use both terms: *event log* and *activity instance log*, to refer to the recorded process information used as starting point; and *event* and *activity instance*, to refer to the recording of an activity execution.

B. Business Process Simulation

A process model is a diagrammatic representation of a process, commonly used to model its control-flow dimension. In this article, we consider process models represented in the Business Process Model and Notation (BPMN). To model the control-flow of a process, a BPMN model consists of *i)* a set of nodes representing the activities of the process; *ii)* a set of gateways modelling the flow of the process, splitting or joining the flow through one or many paths; and *iii)* a set of directed edges relating the activities and gateways between them.

However, a process model can also be extended with information about the execution of the process (e.g. the distribution of activities' processing times) or about the resources that can perform each specific activity. Such enhanced process models are called business process simulation models. Furthermore, in the BPMN formalism, other type of elements can be added to enrich the simulation, such as timer events (i.e. elements that force the process to wait an amount of time before continuing with the simulation). When a BPS model is simulated it produces, based on the information it stores, an event log that replicate the process' behavior.

Table I
RUNNING EXAMPLE: FRACTION OF AN ACTIVITY INSTANCE LOG WITH 10 ACTIVITY INSTANCES STORING THE TRACE (φ), THE EXECUTED ACTIVITY (α), THE START (τ_s) AND END (τ_e) TIMES, AND THE RESOURCE (ρ).

trace	activity	start time	end time	resource
		⋮		
512	Register invoice	10:00:00	10:19:11	BoJack
513	Register invoice	10:19:11	10:34:51	BoJack
514	Register invoice	10:34:51	10:43:12	BoJack
512	Notify acceptance	10:29:11	11:31:45	Carolyn
512	Post invoice	11:15:43	12:08:01	Sarah
513	Post invoice	12:08:01	13:01:09	Sarah
514	Notify acceptance	11:31:45	13:05:36	Carolyn
514	Post invoice	11:31:45	13:16:43	Todd
513	Notify acceptance	13:05:36	13:39:29	Carolyn
512	Pay invoice	18:08:01	18:14:11	BoJack
		⋮		

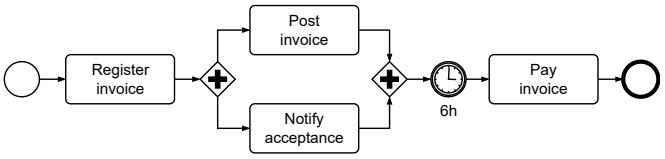

Figure 1. BPS model example formed by 4 activities, two AND gateways, and a timer event, corresponding the event log of Table I.

Figure 1 shows an example of a BPS model corresponding to the event log example from Table I, formed by 4 activities and one timer event. As we can see in Table I, the information about the waiting time is not explicitly present in the event log, but can be represented in the BPS model with a timer event.

C. Related Work

Since the apparition of the first approaches for automated BPS model discovery [2], many techniques have been presented focusing on the enhancement of BPS models to improve the accuracy and precision of the simulation. Camargo et al. presented SIMOD [4], a tool to discover a BPS model from an event log and, using hyperparameter optimization, tune it in order to minimize a loss function and obtain an enhanced BPS model that reproduce more faithfully the process behavior. Following with the aim of enhancing simulation models, Pourbafrani et al. [7] proposed to use coarse-grained BPS models to simulate processes at a higher abstraction level, claiming that it eases the discovery of models mimicking the real behavior, and the exploration of alternative scenarios. Estrada-Torres et al. proposed in [8] a method to discover BPS models in the presence of multitasking, and enhanced with support to resource unavailability. Finally, Meneghello et al. [9] presented a framework to evaluate the accuracy of a BPS model w.r.t. the original event log, and give insights to improve this accuracy. Nevertheless, none of these techniques address the problem of enhancing the BPS model by considering extraneous activity delays.

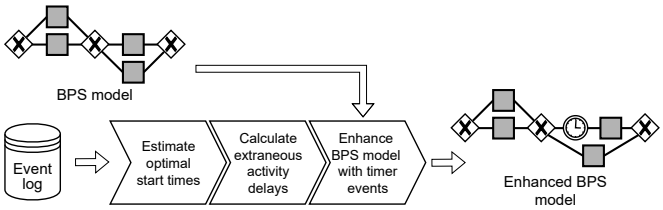

Figure 2. Overview of the approach presented in this paper.

Regarding the presence of waiting times caused by extraneous factors, Camargo et al. find out in [10] that current data-driven BPS models have problems simulating waiting times. The reason behind this is that, currently, BPS models can only reproduce waiting times related to resource contention and resource unavailability, while a great amount of waiting time is, in reality, caused by extraneous factors [5]. Camargo et al. advocate for modeling waiting times by using deep learning simulation models, and compare their performance against data-drive BPS models. Nevertheless, although the results show that deep learning BPS models are more accurate, they cannot be used to perform "what-if" analysis, as these models work as a black box. On the contrary, we aim to produce data-driven BPS models that can be used for "what-if" analysis.

III. Extraneous Activity Delays Modeling

This section describes the proposed approach to enhance a BPS model by adding timer events that capture the waiting times caused by extraneous factors. Figure 2 depicts the main structure of our proposal. With the information recorded in the event log as input, our approach estimates the optimal start times (c.f. Sec. III-A) of each activity instance. Then, combining this value with the start times recorded in the event log, the waiting time associated to extraneous factors is calculated (c.f. Sec. III-B). Finally, for each activity with discovered extraneous delays, a timer event is added previous to it in the BPS model received as input, with the duration distribution result of the observed extraneous delays (c.f. Sec. III-C).

A. Optimal start time estimation

Enablement time: To calculate the extraneous waiting time associated to each activity instance, we need first to identify its enablement time. The *enablement time* (ent) of an activity instance is the instant in time at which it became available for processing. For example, in the event log from Table I, an invoice is first registered, then it is posted and notified as accepted (both in parallel) and, after that, the invoice is paid. In this case, the payment of the invoice cannot be done until it is both posted and notified as accepted. Thus, activity *"Pay invoice"* is not enabled for processing until both activities (*"Post invoice"* and *"Notify acceptance"*) finish.

In a sequential process, an activity instance is enabled when the previous activity instance completes. In other words, the enablement time of an activity instance is the end time of the chronologically preceding activity instance. However, this observation does not hold if there are concurrency relations between activities in a process (a common situation in real-life processes). A concurrency relation between two activities A and B means that these activities can be executed in any order, i.e. sometimes A following B and other times B following A, thus no one enables the other[2]. For example, in Table I, activities *"Post invoice"* and *"Notify acceptance"* are executed in any order (and sometimes even in parallel), as no one is required to start the other.

Hence, in order to precisely measure the enablement time of an activity, we need to detect the concurrency relations between the activities of a process. In order to do this, we propose to use the concurrency oracle of the Heuristics Miner [12]. This method computes a degree of confidence for each relation that it is a concurrent or directly-follows relation, based on the percentage of occurrences of each order. Then, based on a set of defined thresholds, it retrieves the concurrent relations of the process. In this way, once the concurrency relation between the activities is known, the enablement time of an activity instance is set to the end time of the closest previous activity instance not overlapping and not being concurrent to the current one. Accordingly, the enablement time (τ_{ent}) of an activity instance can be defined as follows:

Definition 1 (Enablement Time). Given an activity instance log L, and a concurrency oracle that states if two activities $\alpha_i, \alpha_j \in L$ have ($\alpha_i \parallel \alpha_j$) or do not have ($\alpha_i \nparallel \alpha_j$) a concurrent relation, the *enablement time* of an activity instance $\varepsilon = (\varphi, \alpha, \tau_s, \tau_e, \rho)$, such that $\varepsilon \in L$, is defined as $\tau_{ent}(\varepsilon) = max(\{\tau_e(\varepsilon_i) \mid \varepsilon_i \in L \wedge \varphi(\varepsilon_i) = \varphi(\varepsilon) \wedge \tau_e(\varepsilon_i) < \tau_e(\varepsilon) \wedge \tau_e(\varepsilon_i) \leq \tau_s(\varepsilon) \wedge \alpha(\varepsilon_i) \nparallel \alpha(\varepsilon)\})$, i.e., the largest end time of those activity instances of L, belonging to the same process trace, being previous to ε's end time, previous or similar to ε's start time (i.e. not overlapping), and which activity does not have a concurrent relation with ε's activity.

Figure 3 shows the timeline for one process trace, depicting the enablement relations between the activities. As it can be seen, both *"Post invoice"* and *"Notify acceptance"*, which have a concurrency relation between them, are enabled by *"Register invoice"*. It must be noted that, in this example, the fact that the activities overlap each other already sets their enablement time as the end time of *"Register invoice"*. Nevertheless, the use of the concurrency oracle prevent a wrong calculation in cases where they do not overlap (e.g. trace 513 from Table I). Finally, *"Pay invoice"* is enabled by *"Post invoice"*, its previous non-overlapping activity not being concurrent to it.

[2]We note that other notions of concurrency have been proposed in the field of process mining. An in-depth treatment of concurrency notions in process mining is provided by Armas-Cervantes et al. in [11]. In this paper, we adopt a basic notion of interleaved concurrency between pairs of activities as defined above. The proposed approach is, however, modular and could be adapted to exploit more sophisticated notions of concurrency.

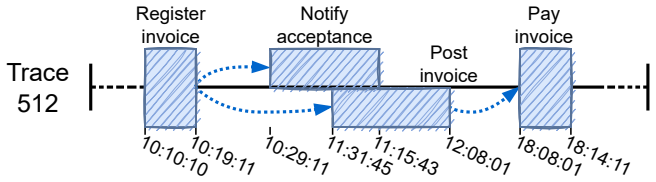

Figure 3. Timeline of trace 512 corresponding to the example in Table I, where each blue box represents each activity instance, and each dotted arrow connects an activity instance (source) with an activity instance it enables (target).

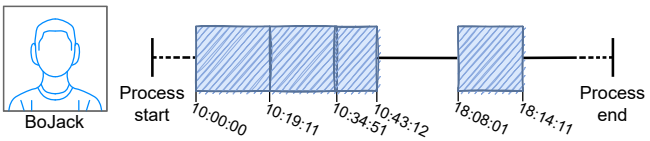

Figure 4. Individual timeline for the resource "BoJack" corresponding to the example in Table I, where each blue box represents each activity instance performed by this resource.

Resource availability time: Once the enablement time of an activity instance is known, its waiting time can be easily computed as the interval from its enablement time until it started being processed. Nevertheless, not all this waiting time is caused by extraneous factors. For example, if there are no resources available, an enabled activity instance must wait until a resource can be assigned to them. To exclude this waiting times associated to resource contention from the extraneous delays, we propose to also calculate the time in which the resource became available to process the activity instance (i.e. the resource availability time).

To develop on this idea, we assume that when a resource ends an activity and becomes available, she/he will start working in the next one, and that there is no multitasking (i.e. the resources only work in one activity at a time). This reasoning is also used in [13] to calculate activity durations in order to check if the workforce of a process has decreased or increased after the pandemic. We propose to build a timeline for each resource by adding all the registered activity instances that she/he performed. Figure 4 shows an example of this timeline. The resource availability time of an activity instance corresponds to the end time of the latest activity instance performed by its resource. Accordingly, the resource availability time (τ_{rat}) of an activity instance is defined as follows:

Definition 2 (Resource Availability Time). Given an activity instance log L, the *resource availability time* of an activity instance $\varepsilon = (\varphi, \alpha, \tau_s, \tau_e, \rho)$, such that $\varepsilon \in L$, is defined as $\tau_{rat}(\varepsilon) = max(\{\tau_e(\varepsilon_i) \mid \varepsilon_i \in L \wedge \rho(\varepsilon_i) = \rho(\varepsilon) \wedge \tau_e(\varepsilon_i) < \tau_e(\varepsilon) \wedge \tau_e(\varepsilon_i) \leq \tau_s(\varepsilon)\})$, i.e., the largest end time of those activity instances of L, processed by the same resource as ε, being previous to its end, and not overlapping.

Optimal start time: In order to discard the waiting times due to resource contention from the extraneous waiting time calculation, we propose to consider the earliest instant at which an activity instance could have started, i.e. its optimal start, instead of its enablement time. We do so by combining its enablement time with the availability of its resource. In this way, we consider that an activity instance could have started as soon as its activity is enabled, and its resource is available to process it. Figure 5 shows an example of this estimation, where the optimal start corresponds to the enablement time. Accordingly, the estimation of an activity instance's optimal start time (τ_{opt}) can be defined as follows:

Definition 3 (Optimal start time). Given an activity instance log L, the *optimal start time* of an activity instance $\varepsilon = (\varphi, \alpha, \tau_s, \tau_e, \rho)$, such that $\varepsilon \in L$, is defined as $\tau_{opt}(\varepsilon) = max(\tau_{rat}(\varepsilon), \tau_{ent}(\varepsilon))$, i.e., the largest of its enablement time and its resource availability time.

B. Extraneous activity delays calculation

Once the optimal start time of all activity instances in the event log is known, their waiting time caused by extraneous factors can be easily computed as the difference between their recorded start time and their optimal start time.

Definition 4 (Extraneous activity delay). Given an activity instance log L of a process with a set of activities A, the *extraneous delay* of an activity instance $\varepsilon = (\varphi, \alpha, \tau_s, \tau_e, \rho)$, such that $\varepsilon \in L$, is defined as $\omega_{extr}(\varepsilon) = \tau_s(\varepsilon) - \tau_{opt}(\varepsilon)$. Accordingly, the set of extraneous waiting times of an activity $\alpha_i \in A$ is defined as $\Omega_{extr}(\alpha_i) = \{\omega_{extr}(\varepsilon) \mid \varepsilon \in L \wedge \alpha(\varepsilon) = \alpha_i \wedge \tau_{ent}(\varepsilon) \neq null\}$, i.e. the set of extraneous waiting times of α_i's activity instances with a valid enabled time.

Figure 5 depicts an example of the estimation of the extraneous delays. It must be noted that, to compute the set of extraneous delays of an activity, we discard its activity instances with no valid enablement time. These cases correspond to two types of activity instances: *i)* the first activity instance of each trace, and *ii)* the activity instances with all predecessors being concurrent or overlapping them. In both cases, the activity instance is enabled by their trace arrival. Martin et al. [14] and Berkenstadt et al. [15] have presented different techniques to estimate the arrival times of process traces that could be used as enablement time for this activity instances. Nevertheless, this is out of the scope of this paper. Therefore, only the delays of those activity instances being enabled by a previous activity instance are used in the estimation of extraneous delays.

C. BPS model enhancement

The final stage of our proposal is to enhance the original BPS model with the information of the discovered extraneous activity delays. To do this, we take all the observed extraneous waiting times previous to each activity, and compute the probabilistic distribution that best fit them. Then, we inject in the BPS model a (durational) timer event previous to each activity with the discovered duration distribution. In this way, the simulation will mimic the extraneous waiting

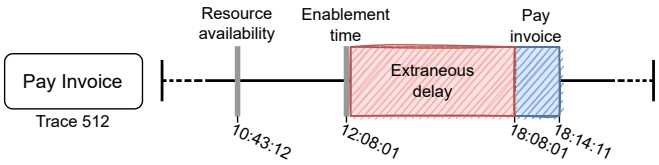

Figure 5. Timeline denoting the start and end times, resource availability, and enablement time for the activity instance "Pay invoice", of the process trace 512, corresponding to the example in Table I. The red striped box denotes the estimated waiting time due to extraneous factors.

times by delaying the execution of the activities based on the timer event that precedes them. It is possible that not all activities in the process present delays caused by extraneous factors, or that the number of delays is scarce w.r.t. their total number of executions. For this reason, we consider an outlier threshold (δ) to inject a timer event only to those activities in which the proportion of positive extraneous delays discovered ($|\{\omega_{extr} \mid \omega_{extr} \in \Omega_{extr}(\alpha) \wedge \omega_{extr} > 0\}| \ / \ |\Omega_{extr}(\alpha)|$) is higher than δ.

Although the presented approach already includes timer events to mimic the extraneous activity delays, the duration of the discovered extraneous waiting times might be smaller than the real ones. This is caused by the interaction between waiting times caused by resource contention and extraneous factors. There are some scenarios in which the waiting time caused by resource contention may eclipse, either partially or totally, the waiting time due to extraneous factors. For example, an enabled activity instance that is waiting due to both extraneous and resource contention factors. Two types of interaction can happen in these cases, *i)* there is extraneous waiting time after the resource finishes the previous activity, so the discovered delay has a duration lower than the real one (partial eclipse), and *ii)* the extraneous waiting time ends before the resource contention, so the activity starts as soon as its resource is available to process it, and no extraneous delay is discovered (total eclipse).

To correct this deviation, we propose to establish a scale factor γ_i for each activity α_i, such that all its discovered extraneous delays are scaled as $\Omega'_{extr}(\alpha_i) = \{\gamma_i \times \omega_{extr} \mid \omega_{extr} \in \Omega_{extr}(\alpha_i)\}$. Then, given a set of $\{\gamma_0, \gamma_1, ..., \gamma n\}$, corresponding to the set of activities $\{\alpha_0, \alpha_1, ..., \alpha_n\} \in A$ with discovered extraneous delays, we launch a Tree-structured Parzen Estimator [16] (TPE) hyperparameter optimization process to obtain the set of γ that leads to a better simulation. To avoid the possibility of overfitting, we split the event log in two partitions, training and validation sets, discovering the extraneous delays from the former, and evaluating each solution in the latter. To perform this split, we order the activity instances by end time and take the first 50% for the train set, and the other 50% for the validation set.[3] Before the start of the optimization process, the Ω_{extr} are estimated using the training set. Then, in each iteration of the optimization, the Ω'_{extr} of each activity are computed based on the set of γ given by the hyperparameter optimization algorithm, and the original BPS model is enhanced with the scaled extraneous delays. To evaluate the iteration, we run 10 simulations of the enhanced BPS model for the same number of traces than the validation set, and compute the mean of the distances between the simulated logs and the validation set.[4] In this way, the TPE hyperparameter optimizer returns the set of γ values that lead to the enhanced BPS model that more closely reflects the temporal behavior of the log.

[3]We perform a strict split considering partial traces to minimize the impact in the training-validation process. If we exclude complete traces, some resources might appear to be available in the training set, when actually they're busy working on a previous activity of a trace that is in the validation set, and that would affect the calculation of the τ_{rat}.

IV. Evaluation

This section reports on an experimental evaluation of the proposed approach to evaluate its accuracy when re-discovering already known extraneous activity delays from a simulated process, and the impact of its enhancement in the simulation of real-life event logs.

With this purpose, two research questions are presented. The first one is related to the accuracy of the extraneous activity delay discovery in simulated scenarios in which they are known. **Research Question 1 (RQ1)**: *Is the proposed technique able to re-discover extraneous activity delays that are known to be present in the process from which the log was produced?* Regarding this research question, we expect that our proposal re-discovers all the extraneous activity delays with variations in their duration due to, as we have explained in Sec. III-C, partial eclipses caused by the resource contention waiting time. The second research question focuses on the impact that including the discovered extraneous activity delays has in the simulation. **Research Question 2 (RQ2)**: *In the context of real-life processes, given an event log and a BPS model of the process, does the proposed technique improve the temporal accuracy of the BPS model?* Regarding this research question, our proposal is enhancing the original BPS model by adding existent waiting times that are not being modeled. For this reason, we expect the enhancement of our proposal to improve the temporal accuracy of the simulation w.r.t. the simulation using the original BPS model.

A. Datasets

In order to evaluate the accuracy of the re-discovery of already known extraneous activity delays (RQ1), we selected a set of five BPS models. Four of these BPS models (with no timer events) are extracted from the examples of the book Fundamentals of Business Process Management [17]: a pharmacy prescription fulfillment process (CVS), an insurance process (INS), a loan application process (LAP), and a procure to pay process (P2P). The other BPS model has been discovered from a confidential real process already containing several message and timer events (CF). For each of the CVS, INS, LAP and P2P processes, we added timer events previous to

[4]We use the Wasserstein Distance (described in Sec. IV-B) between the cycle time distribution as distance measure to minimize by the TPE optimization.

Table II
CHARACTERISTICS OF THE REAL-LIFE EVENT LOGS USED IN THE EVALUATION.

Event log	Traces	Activity instances	Variants	Activities	Resources
BPIC12TR	3,030	16,338	735	6	47
BPIC12TE	2,976	18,568	868	6	53
BPIC17TR	7,402	53,332	1,843	7	105
BPIC17TE	7,376	52,010	1,830	7	113

some activities (in sequential, parallel, and loop scenarios) to mimic the existence of extraneous activity delays. For each dataset, we simulated an event log of 1,000 traces out of the BPS model without timer events, another one with one timer event, and another one with four timer events. Regarding the last process (CF), we simulated one event log of 1,000 traces with the original BPS model, to validate if our proposal could detect the already present timer events.

To evaluate the improvement of our proposal in the temporal accuracy of a BPS model in real-life scenarios (RQ2), we selected the two real life event logs from the Business Process Intelligence Challenges (BPIC) of 2012 [18] and 2017 [19], as they both have start and end timestamps, detailed resource information, and low multitasking. We preprocessed the event log from BPIC 2012 retaining only the events corresponding to activities performed by human resources (i.e. only activity instances that have a duration), and the event log from BPIC 2017 by following the recommendations reported by the winning teams participating in the competition (https://www.win.tue.nl/bpi/doku.php?id=2017:challenge). Then, in order to avoid data leakage in the evaluation of our approach, we split each event log in two sets (the training and the test set). For each dataset, we selected two temporal disjoint intervals (i.e. non-overlapping) with similar trace and event intensity (avoiding the concept change in the activity instances intensity in the BPIC 2017), and retained only the traces that were fully contained in them. In this way, no events in the training set overlap with events in the test set. Table II shows the characteristics of the two training (BPIC12TR and BPIC17TR) and two test (BPIC12TE and BPIC17TE) event logs. To obtain the BPS model to enhance, we used SIMOD [4] to automatically discover a BPS model from the training partitions. These datasets, as well as the results of the evaluation, are publicly available at https://doi.org/10.5281/zenodo.6719731.

B. Experimental setup

We have evaluated both versions of our proposal: the enhancement with the discovered extraneous activity delays (Naive proposal), and the enhancement with the delays obtained after the TPE hyperparameter optimization (TPE Optimized proposal). Both versions were implemented as a Python package, and are publicly available, as well as the code to reproduce the evaluation experiments, in GitHub.[5]

[5]https://github.com/AutomatedProcessImprovement/extraneous-activity-delays/tree/icpm-2022

As configuration parameters of the approach, we used an outlier threshold of $\delta = 0.05$ (only injecting timer events to those activities for which the amount of positive extraneous delays discovered were more than the 5% of observations), a maximum scale factor of $\gamma = 50$, and set 200 iterations for the TPE hyperparameter optimization.

As measures of goodness to evaluate the accuracy in the re-discovery of timer events from the simulated event logs (RQ1), we use precision and recall. In this context, a true positive stands for a discovered timer event that is present in the BPS model from which the simulated log was created, a false positive stands for a discovered timer event that is not present in the BPS model, and a false negative stands for a timer event present in the BPS model that was not discovered. We also report on the Symmetric Mean Absolute Percentage Error (SMAPE) comparing the mean of the discovered delays (forecasted value) with the mean of the distribution of the corresponding timer event (actual value). We use zero as the actual value in the case of false positives, and as the forecasted value in the case of false negatives. The symmetric nature of the SMAPE metric gives a more robust value by fixing the asymmetry of boundlessness, while giving the percentage error allows performing inter-dataset comparisons.

Regarding the real-life event logs (RQ2), we compared the accuracy of the simulations performed using the original BPS model and the BPS model enhanced with our proposal. To do this, we run ten simulations of the same number of traces than the test log with each of the BPS models, and computed the mean of their similarity w.r.t. the test log. In previous work, the similarity between two event logs is made along two dimensions: the control-flow and the temporal dimensions [4], [20]. Given that the injection of timer events does not alter the control-flow of the BPS model, we hereby focus on the temporal dimension. Specifically, we compared each simulated log against the corresponding test log in two ways: *i)* by comparing the distribution of cycle times of their traces; and *ii)* by comparing the distributions of their timestamps (both start and end timestamps). To compare two distributions, we use the Wasserstein Distance (a.k.a. Earth Movers' Distance, or EMD), which measures the amount of movements that have to be applied to one discretized distribution to obtain the other, penalizing the movements by their distance. In this way, a higher EMD value denotes a lower similarity.

The EMD is defined over a pair of discretized distributions (i.e. histograms). To discretize the cycle times, we group the cycle times of the test log into 1000 equidistant bins. Let W be the width between these bins. Each simulated log is then discretized into bins of width W. On the other hand, the distribution of timestamps are discretized as follows. First, we extract all the start and end timestamps in the log, and we group them by date-hour. For example, all timestamps that fall between 27 May 2022 at 10:00:00 and the same day at 10:59:59 are placed in one group (bin). Having discretized the cycle times and the timestamps of the test log and its simulated counterparts, we then calculate their EMD and use this measure to compare the simulation performed with the

original BPS model, with the two variants of our proposal. Importantly, the magnitude of the EMD depends on the dataset (e.g. larger datasets will naturally give rise to larger EMD values). Accordingly, we use the EMD to make intra-dataset comparisons only (i.e. to measure the relative performance of multiple techniques within the same dataset).

C. RQ1 - Accuracy of the extraneous delay discovery

Regarding the results of the re-discovery of already known extraneous activity delays injected as timer events in the BPS model, none of the proposals discovered any non-existent extraneous activity delays in the event logs simulated with no injected timer events (no false positives). Table III depicts the results for the event logs simulated with one and four injected timer events. In those cases, both proposals discovered all the injected timer events (recall of 100%) in all cases except in the P2P process with 4 injected timer events, where both techniques miss one timer event (recall of 75%). This missing timer is explained by one of the activities experiencing a total eclipse in most of its activity instances. Due to resource contention, these activity instances had to wait more for their resource to become available than because of the injected timer event, causing the discovered extraneous waiting time to be zero.

Regarding the duration of the discovered delays, the TPE optimization gets values closer to the real ones in five out of the eight cases (discarding the cases with no injected timer events). Although the TPE optimization leads to delay durations closer to the injected values in five out of the eight cases, the naive proposal is closer in the other three. The reason behind this is that the TPE optimization is learning the delay duration from a subset of the log, and optimizing it by minimizing the distance of the simulated cycle time w.r.t. another subset of the log. This could add noise to the estimation, leading to the observed difference in some cases. Nevertheless, the impact of this difference in the simulation is analyzed in Sec. IV-D.

In the experiments with CF, both versions of our proposal discovered correctly the two timer events, and modeled the six message events as one single timer event. The reasoning behind this is that the six message events are placed at the end of each branch in a six-branch parallel. In this structure, the start of the succeeding activity only happens after the latest message event finishes. As our proposal models the extraneous waiting time as a delay before an activity, the extraneous waiting time occurring after each parallel activity will be modeled as a delay of the succeeding one. For this reason, the approach discovers a single activity delay before the activity being executed after the six message events.

D. RQ2 - Accuracy of the enhanced simulation

Table IV shows the results of the simulation with the (original) BPS model discovered by SIMOD, and the BPS models enhanced with our proposal. As it can be seen, the BPS model enhanced with both versions of our proposal leads, in both datasets, to simulated logs that more closely replicate the temporal dynamics recorded in the original log, relative to the original BPS model. In the case of the naive version of the proposal, the small improvement can be explained by the partial eclipses due to resource contention. As explained before, the resource contention waiting time can partially eclipse the extraneous activity delays' durations, leading to shorter delays and, thus, to shorter cycle times. Nevertheless, the TPE hyperparameter optimizer tunes the delays and obtains an enhanced BPS model that considerably increases the similarity of the simulated event logs.

Table III
PRECISION, RECALL, AND SMAPE FOR THE TIMER EVENT RE-DISCOVERY OF BOTH VERSIONS OF OUR PROPOSAL WITH THE CVS, INS, LAP, AND P2P EVENT LOGS SIMULATED WITH ONE AND FOUR INJECTED TIMER EVENTS.

Dataset	Naive proposal			TPE Optimized proposal		
	Prec.	Recall	SMAPE	Prec.	Recall	SMAPE
CVS 1	1.00	1.00	1.69	1.00	1.00	0.18
INS 1	1.00	1.00	1.95	1.00	1.00	0.13
LAP 1	1.00	1.00	1.53	1.00	1.00	0.81
P2P 1	1.00	1.00	0.96	1.00	1.00	1.57
CVS 4	1.00	1.00	1.43	1.00	1.00	0.40
INS 4	1.00	1.00	1.57	1.00	1.00	1.08
LAP 4	1.00	1.00	0.90	1.00	1.00	1.33
P2P 4	1.00	0.75	1.01	1.00	0.75	1.84

E. Threats to validity

The evaluation reported above is potentially affected by the following threats to the validity. First, regarding *internal validity*, the experiments rely only on five simulated and two real-life processes. The results could be different for other datasets. Second, regarding *construct validity*, we used two measures of goodness based on discretized distributions. The results could be different with other measures, e.g. measures of similarity between time series based on dynamic time warping, which provides an alternative framework for capturing rhythms in time series. Finally, regarding *ecological validity*, the evaluation compares the simulation results against the original log. While this allows us to measure how well the simulation models replicate the as-is process, it does not allow us to assess the goodness of the simulation models in a what-if setting, i.e., predicting the performance of the process after a change.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a technique to enhance a BPS model by modeling the waiting times caused by extraneous factors (i.e. extraneous activity delay) as timer events. The proposal combines the activity's resource and enablement information available in the event log to estimate the extraneous delay of each activity instance, and discover a distribution function of its duration. To improve the estimation, we proposed a TPE hyperparameter optimization process to tune the discovered extraneous activity delays. Then, for each activity with discovered extraneous delays, our proposal adds a timer event previous to it in the BPS model.

Table IV
EMD VALUES (THE LOWER, THE BETTER) BETWEEN THE TEST EVENT LOGS, AND THE EVENT LOGS SIMULATED WITH THE ORIGINAL AND ENHANCED BPS MODELS.

Dataset	Cycle time EMD			Timestamp EMD		
	Original	Naive proposal	TPE optimized proposal	Original	Naive proposal	TPE optimized proposal
BPIC 2012	129.47	120.70	65.80	400.79	393.73	320.66
BPIC 2017	136.85	133.19	20.03	508.20	501.30	344.42

We evaluated the proposal with a set of event logs simulated from BPS models with and without timer events, to mimic already known extraneous activity delays. The goal is to re-discover the injected extraneous activity delays. Our approach re-discovered all the injected extraneous delays except one, and no non-existent ones. With this evaluation, we discovered an interaction between waiting time due to resource contention and the extraneous activity delays. While an activity is being delayed for extraneous reasons, its resource might be working in other tasks. This working time is detected as resource contention, partially eclipsing the extraneous delay and reducing its detected duration. To counter this eclipses, the TPE optimization scales the discovered durations to minimize the distance between the simulated log and the recorded one. Nevertheless, if the resource contention is high enough, the eclipse might be total and no extraneous waiting time is discovered. If this happens in all the instances of an activity, no extraneous activity delay is discovered.

We also evaluated the impact in the simulation of real-life processes, by enhancing already discovered BPS models with two real-life event logs. The evaluation shows that our approach improves the temporal accuracy of the BPS models, producing simulated event logs that more closely mimic the behavior of the process recorded in the original event log.

As future work, we plan to extend our technique by estimating the trace inter-arrival time [14], [15] to discover the extraneous waiting times associated to the first activity of each trace, which are currently being discarded. In addition, we plan to use the resource availability calendars in the BPS model (or discover them if not present), to also consider the resources as unavailable when they are out of their working hours.

One limitation of our technique is that it associates the extraneous waiting time as a delay for the next activity. There may be cases where the extraneous waiting time is caused by the previous activity (e.g. rest time). A different approach, considering the extraneous delays as being dependent on the previous and following activities, might improve the results.

REFERENCES

[1] L. Maruster and N. R. T. P. van Beest, "Redesigning business processes: a methodology based on simulation and process mining techniques," *Knowl. Inf. Syst.*, vol. 21, no. 3, pp. 267–297, 2009.

[2] A. Rozinat, R. S. Mans, M. Song, and W. M. P. van der Aalst, "Discovering simulation models," *Inf. Syst.*, vol. 34, no. 3, pp. 305–327, 2009.

[3] N. Martin, B. Depaire, and A. Caris, "The use of process mining in business process simulation model construction - structuring the field," *Bus. Inf. Syst. Eng.*, vol. 58, no. 1, pp. 73–87, 2016.

[4] M. Camargo, M. Dumas, and O. González, "Automated discovery of business process simulation models from event logs," *Decis. Support Syst.*, vol. 134, p. 113284, 2020.

[5] R. Andrews and M. T. Wynn, "Shelf time analysis in CTP insurance claims processing," in *Workshops on Trends and Applications in Knowledge Discovery and Data Mining (PAKDD 2017)*, ser. LNCS, vol. 10526. Springer, 2017, pp. 151–162.

[6] N. Martin, L. Pufahl, and F. Mannhardt, "Detection of batch activities from event logs," *Inf. Syst.*, vol. 95, p. 101642, 2021.

[7] M. Pourbafrani and W. M. P. van der Aalst, "Extracting process features from event logs to learn coarse-grained simulation models," in *Proceedings of the 33rd International Conference on Advanced Information Systems Engineering (CAiSE 2021)*, ser. LNCS, vol. 12751. Springer, 2021, pp. 125–140.

[8] B. Estrada-Torres, M. Camargo, M. Dumas, L. García-Bañuelos, I. Mahdy, and M. Yerokhin, "Discovering business process simulation models in the presence of multitasking and availability constraints," *Data Knowl. Eng.*, vol. 134, p. 101897, 2021.

[9] F. Meneghello, C. Fracca, M. de Leoni, F. Asnicar, and A. Turco, "A framework to improve the accuracy of process simulation models," in *Proceedings of the 16th International Conference on Research Challenges in Information Science (RCIS 2022)*, ser. LNBIP, vol. 446. Springer, 2022, pp. 142–158.

[10] M. Camargo, M. Dumas, and O. G. Rojas, "Learning accurate business process simulation models from event logs via automated process discovery and deep learning," in *Proceedings of the 34th International Conference on Advanced Information Systems Engineering (CAiSE 2022)*, ser. LNCS, vol. 13295. Springer, 2022, pp. 55–71.

[11] A. Armas-Cervantes, M. Dumas, M. L. Rosa, and A. Maaradji, "Local concurrency detection in business process event logs," *ACM Trans. Internet Techn.*, vol. 19, no. 1, pp. 16:1–16:23, 2019.

[12] A. J. M. M. Weijters and J. T. S. Ribeiro, "Flexible heuristics miner (FHM)," in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*. IEEE, 2011, pp. 310–317.

[13] W.-D. Zabka, P. Blank, and R. Accorsi, "Has the pandemic impacted my workforce's productivity? applying effort mining to identify productivity shifts during COVID-19 lockdown," in *Industry Forum - 19th International Conference on Business Process Management (BPM 2021)*, 2021.

[14] N. Martin, B. Depaire, and A. Caris, "Using event logs to model interarrival times in business process simulation," in *Workshops of the 13th International Conference on Business Process Management (BPM 2015)*, ser. LNBIP, vol. 256. Springer, 2015, pp. 255–267.

[15] G. Berkenstadt, A. Gal, A. Senderovich, R. Shraga, and M. Weidlich, "Queueing inference for process performance analysis with missing lifecycle data," in *Proceedings of the 2nd International Conference on Process Mining (ICPM 2020)*. IEEE, 2020, pp. 57–64.

[16] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS 2011)*, 2011, pp. 2546–2554.

[17] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.

[18] B. van Dongen, "Bpi challenge 2012," Apr 2012. [Online]. Available: https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204/1

[19] ——, "Bpi challenge 2017," Feb 2017. [Online]. Available: https://data.4tu.nl/articles/dataset/BPI_Challenge_2017/12696884/1

[20] M. Camargo, M. Dumas, and O. G. Rojas, "Discovering generative models from event logs: data-driven simulation vs deep learning," *PeerJ Comput. Sci.*, vol. 7, p. e577, 2021.