dynamik: A Tool for Performance Drift Detection in Business Processes

Victor Gallego-Fontenla[1,*], Frederik Milani[1] and Marlon Dumas[1]

[1]*University of Tartu, Tartu, Estonia*

Abstract

Business processes are often subject to unplanned changes that affect their performance, also known as performance drifts. For example, some resources being reallocated to different teams, or changes in batching policies, may result in higher cycle times. This paper presents *dynamik*: a Web-based tool that analyzes event logs of business processes to detect drifts in the cycle time of a process, and to identify possible causes of each detected drift. *dynamik* is based on a collection of factors that may affect cycle time, derived from previous work on resource modeling and waiting time analysis. Statistical testing is used to determine which of these factors are significantly different before and after each observed performance drift.

Keywords

business process change, process performance, performance drift, cycle time

Metadata description	Value
Tool name	dynamik
Current version	1.0
Legal code license	Apache 2.0
Languages, tools and services used	Python, React, Remix, RabbitMQ
Supported operating environment	Google Chrome 127+, Microsoft Edge 127+, Firefox 127+, Opera 112+, Safari 17.4+
Download/Demo URL	http://dynamik.cloud.ut.ee
Documentation URL	https://github.com/AutomatedProcessImprovement/dynamik-demo/blob/main/README.md
Source code repository	https://github.com/AutomatedProcessImprovement/dynamik-demo
Screencast video	https://youtu.be/zqWbfaXtwWs

1. Significance and Innovation

Business processes are prone to change, be it as a result of planned process redesign [1], or due to external factors (e.g. increase in customer demand) or changes in resource behavior [2, 3]. Examples of unplanned changes include resources adapting the way they perform activities in response to workload variations, resources being replaced by new ones who bring in different work practices, workarounds being put in place to deal with new types of cases, or changes in

ICPM 2024 Tool Demonstration Track, October 14-18, 2024, Kongens Lyngby, Denmark

*Corresponding author.

victor.gallego@ut.ee (V. Gallego-Fontenla); fredrik.milani@ut.ee (F. Milani); marlon.dumas@ut.ee (M. Dumas)

0000-0002-4149-919X (V. Gallego-Fontenla); 0000-0002-1322-915X (F. Milani); 0000-0002-9247-7476 (M. Dumas)

the ratio between different case types [4]. From a tactical management perspective, unplanned changes are particularly relevant when they affect key performance metrics, such as labor cost or cycle time. When unplanned changes lead to such *performance drifts*, analysts and managers need to understand the root causes of these drifts and implement corrective actions.

The *dynamik* tool analyzes event logs of business processes to detect performance drifts and to provide a list of possible causes of each detected drift. Specifically, *dynamik* focuses on detecting and explaining drifts in cycle time. *dynamik* relies on a decomposition of cycle time into factors, summarized in Table 1. This decomposition starts from the observation that a change in cycle time may come from changes in processing time or changes in waiting time. In turn, changes in processing time may stem from changes in resource behavior, or changes in the distribution or nature of the activities in the process. To model changes in resources behavior, *dynamik* relies on the concept of resource profile [5], which it extends with a dual concept of *activity profile*. On the waiting time front, *dynamik* relies on the waiting time causes identified in [6], viz. resource contention, resource availability, prioritization, and batching. Resource contention is modeled using notions from queuing theory, viz. arrival rate and service rate. Thus, *dynamik* consolidates a range of previous work in the field of resource behavior and waiting time analysis, into an integrated tool for performance drift detection.

2. Architecture

dynamik is a Web application (cf. Figure 1) with a responsive Web frontend, and a coordinator-worker backend architecture. This backend architecture enables the tool to scale horizontally (by adding worker nodes) and to provide some level of error resilience (e.g. in case of temporary failures on the coordinator or worker nodes).

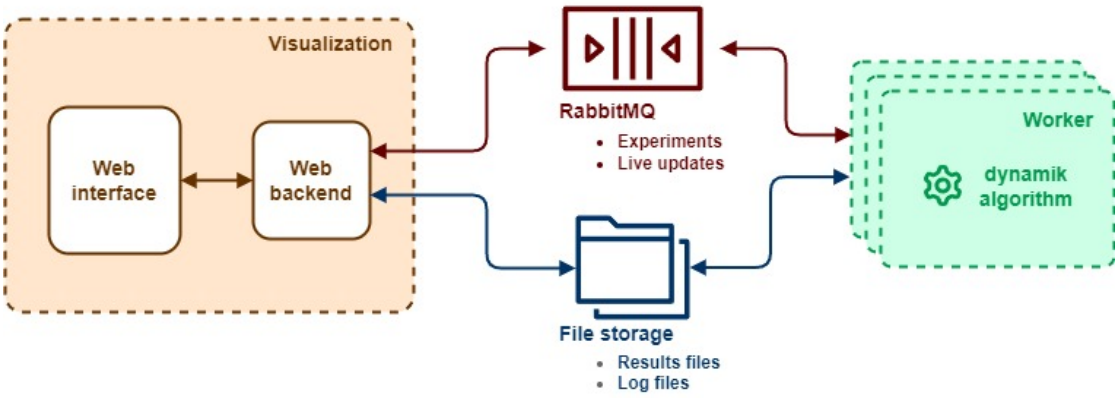

Figure 1: Overview of the *dynamik* tool architecture.

Users interact with the Web frontend to upload log files and to submit requests to detect and analyze performance drifts. Each request is called an *experiment*. The coordinator (in the Web

Table 1
Description of the change factors supported by *dynamik*.

Factor	Factor description
Resource profile	
Utilization index	The utilization index quantifies a resource's activity level within a process. It is calculated as the percentage of time the resource is actively engaged in process tasks relative to its total availability.
Performance deviation	Performance deviation measures the extent to which a resource's activity execution time deviates from the average.
Collaboration index	The collaboration index measures how frequently two resources work in the same process instance.
Circadian effort distribution	The circadian effort distribution shows the relative frequency with which resources finish activity instances for each hour of the week.
Work distribution	The work distribution shows the amount of workload put into executing each activity.
Activity profile	
Activity frequency	The activity frequency measures how frequently an activity is executed during a period of time.
Circadian arrival distribution	The circadian arrival distribution, which measures the distribution of arrivals of activity instances over the week.
Co-occurrence index	The co-occurrence index measures the number of cases two activities appear in the same process instance.
Complexity deviation	The complexity deviation measures the time it takes to execute one specific activity with respect to the average time it takes to execute all activities.
Effort demand	The effort demand measures which percentage of the total processing time is dedicated to execute one specific activity.
Resource calendars	Resource calendars capture the availability schedules for the resources, showing when a resource is *on-* or *off-dutty* during the week, with a hourly granularity.
Arrival rate	The arrival rate describes how fast new cases arrive at the system. The unit is the number of new cases per hour.
Service rate	The service rate describes how fast cases are completed in the process. The unit is the number of cases completed per hour.
Prioritization policies	Prioritization policies are a set of rules that determine when an activity instance should be executed out of the standard FIFO order. These rules can consider both mandatory activity instance attributes like activity name and domain-specific attributes such as client tier or country of origin.
Batching policies	Batching policies define when groups of activity instances are executed together instead of individually upon enablement. These policies can be organized into a two-level hierarchy: 1. Batch creation policies: Determine which activity instances are grouped into batches. Common factors include activity name, but also domain-specific attributes. 2. Batch firing policies: Specify when a batch is executed. Typical rules consider batch size, maximum waiting time for instances within the batch, or absolute time restrictions (e.g., executing every Sunday at 12:00).

backend) decomposes each incoming experiment into a set of *jobs*. Examples of jobs include detecting all performance drifts in a log, checking if a given drift may be due to changes in batching policies, or checking if a drift may be due to changes in the case arrival rate. The coordinator publishes the jobs to be performed in a message queue. It then periodically checks for events indicating the completion of a job (called *live updates*) in another message queue. A pool of workers continuously monitors the queue of inbound experiments. Upon receiving a job, a worker consumes it and executes the associated drift detection or analysis script.

dynamik finds performance drifts using a sliding window approach. It slides a cursor across the event log timeframe. At each time point, it tests if the distribution of cycle times in the window preceding this time point is statistically different from the cycle time distribution in the succeeding window. To avoid false positives, the detection of the drift is delayed until it has been confirmed for several consecutive time points, determined by a parameter. For each causal factor in Table 1, *dynamik* then applies statistical testing to determine if there is a significant difference in this factor in the preceding and the succeeding time windows. Each such factor is then reported back as a possible cause.

Upon completion, the worker writes the results of the job into a file storage and publishes a live update in the corresponding message queue. When the coordinator in the Web backend detects a new live update, it fetches the corresponding results from the storage and pushes it back to the Web frontend. The Web frontend is updated every time it receives a result (e.g. whenever it receives an update with an additional potential cause of a performance drift).

The components of *dynamik* are outlined below:

The Web application: providing a user interface for event log upload, experiment creation, and result visualization.

The *dynamik* workers: A set of Python processes that execute the performance drift detection algorithm, and the various change analysis algorithms (for each of the factors listed in Table 1).

The message broker: Implemented using RabbitMQ, this queue facilitates asynchronous communication between the web application and the workers, enabling scalable and decoupled system components.

The shared file storage: A shared folder where logs and experiment results are saved for long-term retention and analysis, where both the interface and the workers have access.

3. Functionality

Below, we present the main functionalities of *dynamik*. Specifically, we focus on activity instance log upload, experiment configuration, results overview and change details.

Activity instance log upload: *dynamik* requires a CSV file containing an activity instance log, with a maximum size of 50MiB. Users can upload files through a dialog or drag-and-drop. The file must include columns for case, activity, resource identifiers, and start/end timestamps. To calculate waiting times, *dynamik* leverages the enablement timestamp,

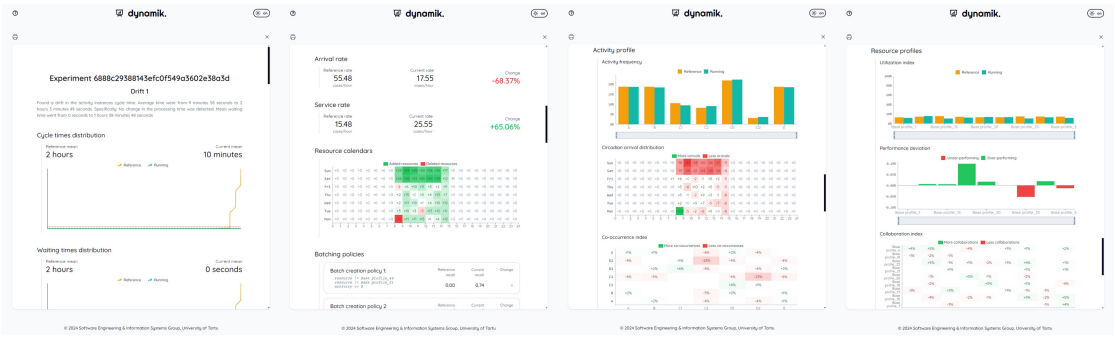

Figure 2: Drift details screen in *dynamik*.

indicating when an activity becomes executable from a control flow perspective. While this data is often missing from the log, *dynamik* can automatically compute it using the Split Miner *concurrency oracle* [7]. By analyzing execution traces, the concurrency oracle identifies concurrent activities within the process. With this information, *dynamik* assigns the enablement timestamp for each activity instance to the end timestamp of the immediately preceding non-concurrent activity instance. If no such preceding activity is found, the enablement timestamp is set to the start timestamp, resulting in a zero waiting time for that instance.

Experiment configuration: Following log file upload, users configure the experiment using a guided wizard. The first step involves mapping CSV columns to activity instance attributes. The next step focuses on drift detection algorithm parameters: window size (determining the reference model and current behavior timespan), drift magnitude (minimum difference for drift detection), and number of warnings before confirming a definitive drift. Larger window sizes detect larger changes, while drift magnitude allows for fine-tuning sensitivity. The number of warnings controls the duration of a drift before notification, filtering out temporary changes. The final wizard step allows users to review and confirm the configuration. If necessary, they can go back and make changes.

Results overview: Once a request is submitted, *dynamik* provides an interface with the execution status and the results overview. This interface is updated incrementally, every time a drift or a cause of a drift is detected. As the experiment identifier is unique and invariant, the user can save the URL so they can access the drift detection results later, or they can share the URL with others.

Change details: The last view provided by *dynamik* is the change details screen (Figure 2). This screen shows the details of each potential drift cause, using a set of plots. The tool uses line plots for the time distributions, bar plots for frequency distributions, heat maps for circadian distributions and relations between pairs, and plain text for simple indicators such as the arrival or service rate or the batching and prioritization policies.

4. Maturity & Availability

We have tested *dynamik* on real-life logs, including the logs of the 2012 and 2017 Business Process Intelligence Challenge.[1] The tests have shown that *dynamik* can handle event logs with up to around 500K activity instances. The tool identifies performance drifts that appear to be relevant, although they have not yet been validated with domain experts. We have also tested *dynamik* on synthetic logs containing explicitly injected performance drifts (with known causes). In this setting, *dynamik* achieved a precision of 67%. However, the recall is low (33%), meaning that it misses possible causes of drift, particularly when the effects of these missed causes are "hidden" by the causes that are correctly identified. In future work, we plan to evaluate the tool's usefulness and usability with end users and to evolve it based on user feedback.

The links to access the tool and the source code, and the link to a video demonstration, are provided in the metadata table of this paper. The activity instance log used in the video demonstration is available at: https://owncloud.ut.ee/owncloud/s/DCE2kQ4TY5si2nJ. The example result shown in the video can be accessed at: http://dynamik.cloud.ut.ee/results/demo.

Acknowledgments

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

References

[1] M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede (Eds.), Process-Aware Information Systems: Bridging People and Software Through Process Technology, Wiley, 2005. doi:10.1002/0471741442.

[2] B. Weber, M. Reichert, S. Rinderle-Ma, Change Patterns and Change Support Features - Enhancing Flexibility in Process-aware Information Systems, Data and Knowledge Engineering 66 (2008) 438–466. doi:10.1016/j.datak.2008.05.001.

[3] R. P. J. C. Bose, W. M. P. van der Aalst, I. Zliobaite, M. Pechenizkiy, Dealing With Concept Drifts in Process Mining, IEEE Transactions on Neural Networks and Learning Systems 25 (2014) 154–171. doi:10.1109/TNNLS.2013.2278313.

[4] W. M. P. van der Aalst, Process Mining - Data Science in Action, Second Edition, Springer, 2016. doi:10.1007/978-3-662-49851-4.

[5] A. Pika, M. Leyer, M. T. Wynn, C. J. Fidge, A. H. M. ter Hofstede, W. M. P. van der Aalst, Mining Resource Profiles from Event Logs, ACM Transactions on Management Information Systems 8 (2017) 1:1–1:30. doi:10.1145/3041218.

[6] K. Lashkevich, F. Milani, D. Chapela-Campa, I. Suvorau, M. Dumas, Unveiling the causes of waiting time in business processes from event logs, Information Systems 126 (2024) 102434. doi:https://doi.org/10.1016/j.is.2024.102434.

[7] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, A. Polyvyanyy, Split Miner: Automated Discovery of Accurate and Simple Business Process Models from Event Logs, Knowledge and Information Systems 59 (2019) 251–284. doi:10.1007/s10115-018-1214-x.

[1] https://www.tf-pm.org/competitions-awards/bpi-challenge