Ocelescope: An Extensible Web Framework for Object-Centric Process Mining

Görkem-Emre Öztürk, Nina Graves, Aaron Küsters, Sebastian Menne and Wil M. P. van der Aalst

Chair of Process and Data Science, RWTH Aachen University, Ahornstraße 55, 52072 Aachen

Abstract

Object-centric process mining has led to a growing ecosystem of research tools, many of which reimplement common functionality and are difficult to reuse, extend, or maintain. We present *Ocelescope*, an extensible, open-source web framework that provides commonly needed functionality for object-centric process mining out of the box, including data handling, filtering, and visualization. Ocelescope supports two extension mechanisms: a plugin system for turning research prototypes into dynamically loadable plugins with automatically generated inputs and visualized results, and a module system for building more advanced process mining tools on top of Ocelescope while reusing its existing functionality.

Keywords

Process Mining, Object-Centric Event Data, Extensible Software

1. Introduction

Object-centric event data enables the analysis of interactions between multiple business objects within a system of processes, providing a more accurate and holistic view of complex real-world behavior. Since the introduction of object-centric process mining (OCPM), a number of tools have been developed to support its use, spanning a variety of formats, including standalone Python scripts (e.g., *TOTem* [1], *Super Variants* [2]), web-based applications (e.g., *OCπ* [3], *OCPQ* [4], *ProAct* [5]), and ProM plugins (e.g., *Object-Centric Local Process Models* [6]). While the growing tool landscape demonstrates an increasing interest in OCPM, it also reveals significant practical limitations. Most tools use old versions of programming languages, heavily depend on outdated practices and libraries, and have little to no coding standards [7]. Often, these tools are the result of one-time projects to provide a proof of concept. Consequently, they are often difficult to install and set up, leaving users to infer required dependencies or resolve build errors. These quality issues are understandable given researchers' focus, but they hinder reproducibility and slow the development of new approaches. Researchers are often required to reimplement existing approaches or basic functionalities, such as log filtering and visualization. The following commonalities were observed in the current tool landscape [7]: i) Python and Rust are often the programming languages of choice (e.g., *TOTem* [1], *Super Variants* [2], *OCPQ* [4]), ii) many tools are web applications using frontend frameworks such as React or Angular to allow for easier user interaction (e.g., *OCπ* [3], *OCPQ* [4], *ProAct* [5]), and iii) many tools require and use similar functionalities and elements such as log imports and exports, filtering, and graph-based visualizations (e.g., *OCπ* [3], *OCPQ* [4], *ProAct* [5]).

In this work, we present *Ocelescope*, an open-source web framework for simple implementation and deployment of new OCPM approaches. It provides frequently required functionalities out of the box as illustrated in Figure 1. Ocelescope is designed to be extensible by providing two complementary extension mechanisms. Plugins are Python-based extensions that are loaded dynamically at runtime and allow researchers to turn prototypes into usable tool functionality with automatically generated user interfaces and results that can be visualized and exchanged within the framework. Modules are intended

Proceedings of the BPM 2026 Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Forum (BPM 2026) , Toronto, Canada, September 27th to October 2nd, 2026.

goerkem.oeztuerk@rwth-aachen.de (G. Öztürk); graves@pads.rwth-aachen.de (N. Graves)

0009-0002-5760-924X (N. Graves); 0009-0006-9195-5380 (A. Küsters); 0000-0002-0955-6940 (W. M. P. van der Aalst)

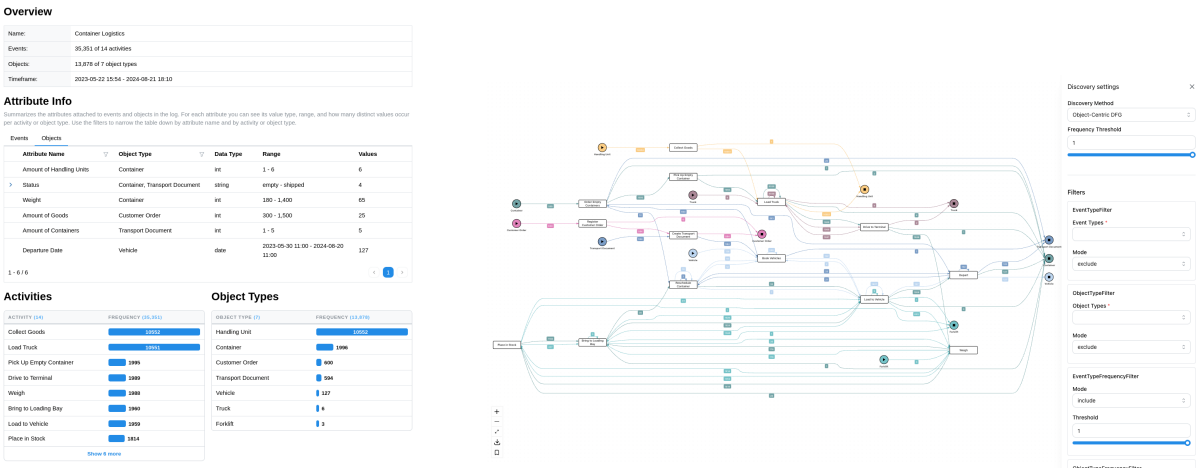


Figure 1: Built-in functionality of Ocelescope. The framework supports the inspection and exploration of OCELs (left) as well as an interactive discovery and visualization of object-centric process models (right).

for developers who want to build specialized OCPM applications on top of Ocelescope while reusing its existing functionality. Through defined extension interfaces, modules can add new React-based views to the frontend and integrate new process mining functionality in Python.

2. Ocelescope Tool

Ocelescope runs locally using Docker and can be started easily with a Docker Compose script. The Ocelescope base tool offers common functionalities including the import, export, inspection, and filtering of object-centric event logs (OCELs), as well as the discovery and visualization of process models. It can be distributed through container registries, ensuring a consistent environment across installations and allowing plugins and modules to run without additional setup[1]. It consists of a TypeScript-based frontend built with Next.js and a Python backend built with FastAPI. The built-in functionality and both extension mechanisms operate on OCELs which can be imported and exported in the OCEL 2.0 standard [8] in all exchange formats. Case-centric logs in XES standard can also be imported/exported; they are treated as OCELs with a single object type. Apart from event logs, process mining artifacts, *resources* (similar to those used in ProM [9]), can be imported, exchanged and exported. Ocelescope provides predefined resource types for common artifacts such as object-centric Petri nets and directly-follows graphs, while plugins and modules can define custom resource types through Python classes. The base tool, modules and plugins share resources as outputs and inputs via a JSON format as a common representation, and resource types that include a specification for visualization can be rendered automatically using built-in visualization types such as graphs, tables, and charts.

Plugins. Plugins offer a lightweight way to run custom process mining functions inside Ocelescope: implemented as Python classes, they are easy to write even for users with mainly scripting experience, and their methods can be executed directly inside the tool. Ocelescope generates the frontend input form automatically from the method's function signature and, for additional settings, from an optional Pydantic configuration class – developers never write frontend code. Figure 2 illustrates how a plugin definition is connected to an automatically generated user interface within the framework. Parameters such as event logs and resources are recognized automatically and offered as selectable inputs from those currently available in Ocelescope. Configuration classes can also define context-aware fields, e.g.,

[1]project site (incl. documentation, guide, and plugin library): https://www.ocelescope.org/, source code: https://github.com/promi4s/ocelescope, demo video: https://www.ocelescope.org/getting-started/ocelescope-introduction/

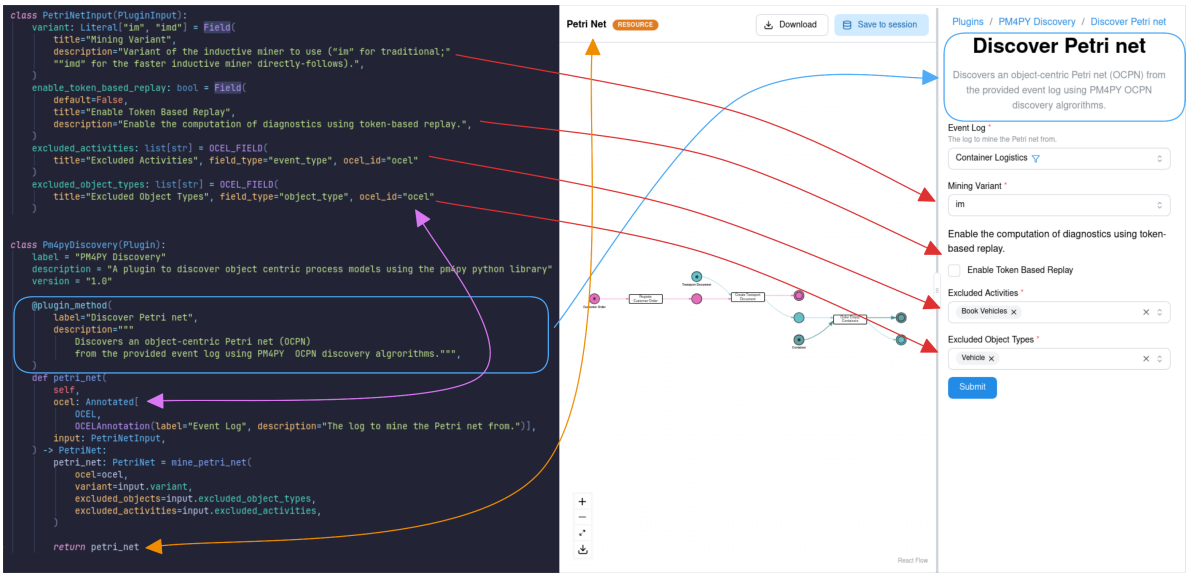


Figure 2: Example of a plugin in Ocelescope. The user interface (left) is automatically generated from the Python plugin definition (right). The event log is derived from the function's parameters, while additional input fields such as the mining variant and filters are defined in a Pydantic configuration class. Context-aware fields like activity and object type selectors are populated based on the selected event log. After execution, the output, a discovered object-centric Petri net, is displayed in the results section and can be inspected, shared or exported.

a dropdown of activity or object types drawn from the selected log. Plugin runs execute asynchronously and in parallel, keeping the interface responsive. Returned event logs and resources are displayed automatically in the plugin interface, with built-in visualizations rendered directly. These outputs are then available across the framework: they can be inspected and exported, and reused as inputs in (other) plugins and modules. Plugins are packaged as `.zip` files, which makes them easy to share, and a new plugin becomes available immediately after upload without rebuilding or restarting the application. They are limited to the packages included in the Ocelescope build, though functionality implemented in other languages, such as Rust, can still be integrated via Python bindings.

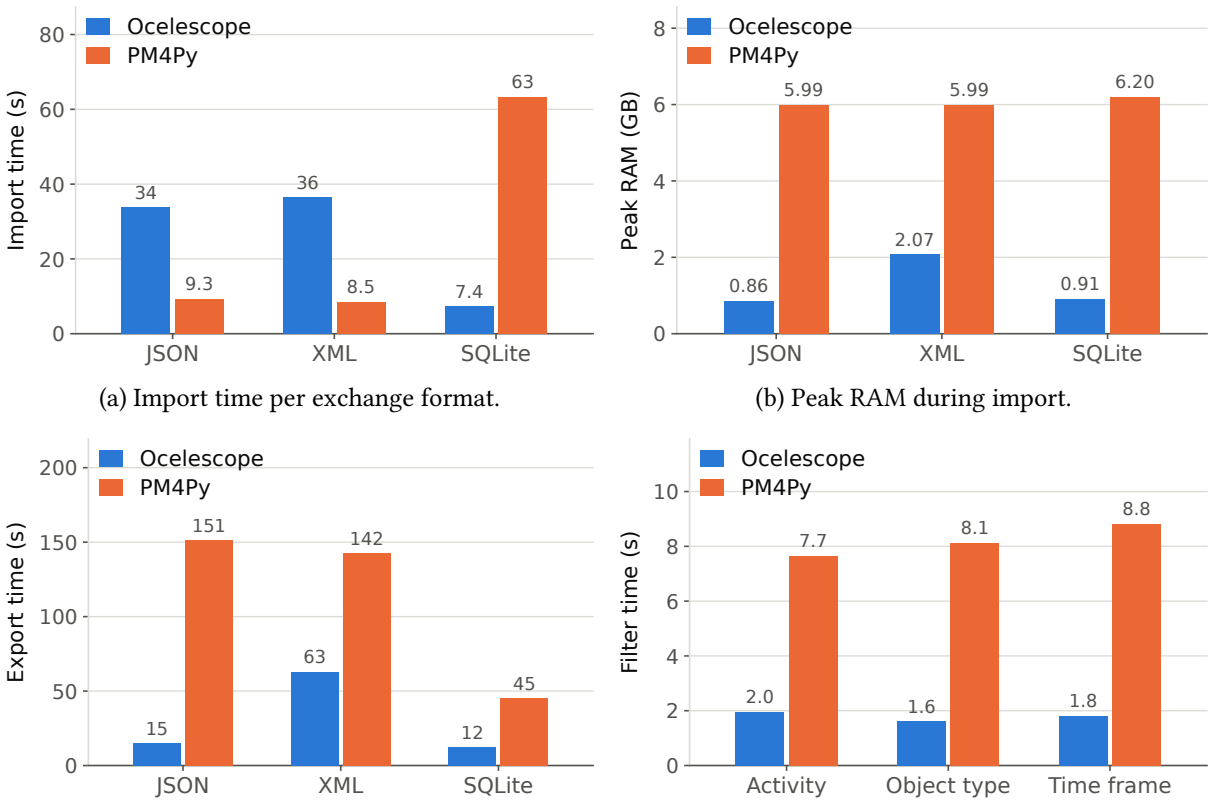
Modules. Modules are intended for more specialized OCPM tools with custom views and more interactive functionality on top of Ocelescope. While plugins mainly follow an input-output pattern, modules allow researchers to implement custom React-based views and the corresponding backend functionality for process mining tasks in Python. Hence, only the new approach must be implemented while common functionalities such as log handling, filtering, or visualization can be leveraged. Modules can also be shared and reused across different tools using the Ocelescope framework.

3. Conclusion

Ocelescope addresses recurring challenges in OCPM tool development by providing shared functionality – log handling, filtering, visualization – out of the box, reducing redundant effort across tools. It supports extensibility via plugins (lightweight Python extensions) and modules (full frontend/backend control for specialized tools), making it easier to prototype, reuse, and share new approaches without building standalone applications. Ocelescope is open-source, runs locally via Docker, and integrates with libraries such as PM4Py.

Performance. Ocelescope stores every OCEL as a local DuckDB[2] file, so users can work on several logs at once without loading them into memory. The base tool and its extensions access these logs

[2] https://duckdb.org/



(a) Import time per exchange format.

(b) Peak RAM during import.

(c) Export into each exchange format.

(d) Filtering by activity, object type, and time frame.

Figure 3: Library-level benchmarks of `ocelescope` against `pm4py` on the *AOE2* log [11] (2.4 M events; 851 MB as SQLite, 1.21 GB as XML, 1.53 GB as JSON), reporting the median of three runs on an Intel Core i7-1260P with 32 GB RAM. Peak RAM is measured on top of the interpreter baseline.

through the `ocelescope` library[3], which lets them either query the underlying DuckDB database directly with SQL or work on data structures equivalent to those of other Python process mining libraries such as `pm4py` [10]. The library ships its own importers and exporters that stream OCEL files into and out of DuckDB instead of materialising the whole log in memory. As shown in Figures 3a and 3b, this trades import speed for a smaller memory footprint in the text-based JSON and XML formats, whereas for SQLite the streaming importer is both faster and more memory efficient than `pm4py`. On the export side, `ocelescope` is faster than `pm4py` in all three formats (Figure 3c). Overall, this approach costs little performance, while some operations, such as filtering, are even faster than their equivalents in `pm4py` (Figure 3d). Figure 3 reports an excerpt of the benchmark suite; the full results, covering further logs, formats, and operations, are published on the project site[4].

Maturity. The usability of both plugin development and tool usage of a beta version have been evaluated in user studies with 8 and 13 participants via case studies and a standardised TAM questionnaire. The results showed particularly high scores for perceived ease for both usage and plugin development. Concrete suggestions for improvement, collected via a free-text field, mainly addressed a request for more interactive feedback, especially for plugin runs. [7] The current version of Ocelescope provides a stable foundation with full support for plugins and modules, including more usability features such as a more interactive interface for plugin runs. Furthermore, the project site also includes basic documentation on the tool, API-references for the library, detailed guides for using Ocelescope and developing plugins (including a tutorial) as well as a plugin library with the first seven plugins. Processes for submitting plugins, reporting bugs, requesting features and additional dependencies are also set up via

[3] https://pypi.org/project/ocelescope/

[4] https://www.ocelescope.org/ocelescope/benchmark/

the project site and GitHub.

Future Work. Planned extensions include more predefined resource types, richer input forms and additional visualization options. Further base functionalities for inspecting and editing event logs as well as more detailed instructions on the development of modules are currently in progress.

Acknowledgments

The project on which this work is partially based upon was funded by the German Federal Ministry of Research, Technology and Space under grant number 16IS26019. Partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC-2023 Internet of Production - 390621612.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT (GPT-5.4) in order to: grammar and spelling check. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] L. Liss, J. N. Adams, W. M. P. van der Aalst, TOTeM: Temporal Object Type Model for Object-Centric Process Mining, in: Business Process Management Forum, Springer Nature Switzerland, 2024. doi:10.1007/978-3-031-70418-5_7.

[2] J. N. Adams, E. Hastrup-Kiil, G. Park, W. M. P. van der Aalst, Super Variants, in: Business Process Management, Springer Nature Switzerland, 2024. doi:10.1007/978-3-031-70396-6_7.

[3] J. N. Adams, W. M. P. van der Aalst, OCpi: Object-Centric Process Insights, in: Application and Theory of Petri Nets and Concurrency, Springer International Publishing, 2022. doi:10.1007/978-3-031-06653-5_8.

[4] A. Küsters, W. M. P. van der Aalst, OCPQ: Object-Centric Process Querying and Constraints, in: Research Challenges in Information Science, Springer Nature Switzerland, 2025. doi:10.1007/978-3-031-92474-3_23.

[5] G. Park, W. M. van der Aalst, Operational process monitoring: An object-centric approach (2025). doi:10.1016/j.compind.2024.104170.

[6] V. Peeva, M. Porsil, W. M. P. van der Aalst, Object-Centric Local Process Models, in: Process Mining Workshops, Springer Nature Switzerland, 2025. doi:10.1007/978-3-031-82225-4_28.

[7] G.-E. Öztürk, Ocelescope: An extensible web framework for object-centric process mining, 2025. URL: https://www.pads.rwth-aachen.de/cms/pads/studium/Abgeschlossene-Abschlussarbeiten/2026/~btrlrh/Bachelor-Thesis-Ocelescope-An-Extensi/lidx/1/, bachelor thesis at Chair of Process and Data Science, RWTH Aachen University.

[8] A. Berti, I. Koren, J. N. Adams, G. Park, B. Knopp, N. Graves, M. Rafiei, L. Li ß, L. T. G. Unterberg, Y. Zhang, C. Schwanen, M. Pegoraro, W. M. P. van der Aalst, OCEL (Object-Centric Event Log) 2.0 Specification, 2024. doi:10.48550/ARXIV.2403.01975.

[9] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, W. M. P. van der Aalst, The ProM Framework: A New Era in Process Mining Tool Support, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005. doi:10.1007/11494744_25.

[10] A. Berti, S. Van Zelst, D. Schuster, PM4Py: A process mining library for Python, Software Impacts (2023). doi:10.1016/j.simpa.2023.100556.

[11] L. Liss, N. Elbert, C. M. Flath, W. M. P. van der Aalst, Object-Centric Event Log for Age of Empires Game Interactions (Version 2), 2024. doi:10.5281/ZENODO.13365584.