Easy and Efficient Object-Centric Process Querying with the OCPQ Tool

Aaron Küsters[1], Wil M.P. van der Aalst[1]

[1]Chair of Process and Data Science (PADS), RWTH Aachen University, Germany

Abstract

OCPQ is a framework and graphical tool for querying object-centric process data. Traditional process querying relies on case-centric event data, which often cannot represent the multiple interacting objects and perspectives of real-life processes accurately. Querying individual events or cases is no longer sufficient for object-centric data. The OCPQ tool allows more flexible queries for arbitrary combinations of objects and events. Queries are represented graphically as node trees in the OCPQ tool, enabling constructing complex queries visually and without programming experience. The query backend of the tool is implemented in the Rust programming language with a focus on fast execution.

Keywords

Process Mining, Object-Centric Event Data, Process Querying

1. Introduction

Process Querying research covers various methods, including retrieving and manipulating process models and data stored in process repositories [1]. In this demo paper, we focus on the retrieval of information and process instances from stored event data. For traditional event data, there are a few different approaches, like the *Process Instance Query Language* (PIQL) [2] and the *Celonis Process Querying Language* (Celonis PQL) [3], that allow retrieving cases or events satisfying specified criteria. For object-centric data, these approaches are not directly applicable due to the lack of a clear case notion. In [4], the authors present an approach for storing and retrieving object-centric event data in a graph database using Neo4j and the Cypher querying language. Similarly, SQL queries can be used to retrieve information from object-centric event data stored in SQL databases, as specified in [5]. However, general querying languages, like SQL or Cypher, are often not a good fit for the types of queries needed to explore process data, and are largely unapproachable for users without programming experience.

Our *Object-Centric Process Querying* approach, *OCPQ*, as presented in [6], proposes a novel, highly expressive querying solution for object-centric process data. Inspired by the classical Z Notation [7], subqueries are modeled as extensions of the parent queries, with potentially additional queried entities or filters on top. As such, even more complex queries can be translated to a tree of simpler query nodes. Notably, the OCPQ method allows querying arbitrary

Proceedings of the Best BPM Dissertation Award, Doctoral Consortium, and Demonstrations & Resources Forum co-located with 23rd International Conference on Business Process Management (BPM 2025), Seville, Spain, August 31st to September 5th, 2025.

kuesters@pads.rwth-aachen.de (A. Küsters); wvdaalst@pads.rwth-aachen.de (W. M.P. van der Aalst)

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

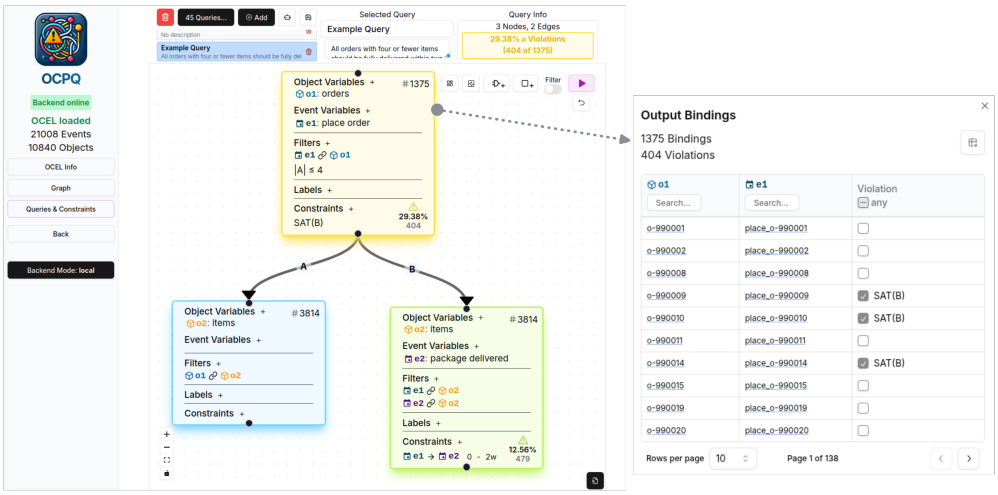

Figure 1: An example OCPQ constraint: All orders with four or fewer items should be fully delivered within two weeks after placing the order.

combinations of objects and events, so-called *bindings*, by specifying their object and event types, and any additional filter predicates, e.g., that there should be event-to-object (E2O) or object-to-object (O2O) relationships between them. In this demo paper, we present the OCPQ tool implementation in more detail, including new features like general binding annotations, OCEL filtering, or early-stopping on overload. The OCPQ tool allows visually modeling and evaluating object-centric process queries on a loaded object-centric dataset. As the result of each query or subquery, an output table can be explored, where individual entries correspond to bindings, i.e., combinations of event or object instances, as defined by the query. Each binding row can additionally be annotated with additional labels or information, for example a violation indicator or a key performance indicator (KPI) value. Figure 1 shows an example OCPQ query with integrated constraints. The graphical query is shown on the left and the output table of the root node is shown on the right.

The remainder of this paper is structured as follows: First, we present an overview of the OCPQ tool and its capabilities in Section 2. Next, in Section 3, we provide a brief description of the OCPQ tool's implementation. Finally, we conclude this paper in Section 4, also giving an outlook on future work.

2. OCPQ Tool

The OCPQ tool is available for download at https://ocpq.aarkue.eu, where the documentation can also be found. The source code of OCPQ is publicly available at https://github.com/aarkue/ocpq. A short demo video of the tool is available at https://github.com/aarkue/ocpq-demo.

Initially, an object-centric event log has to be imported. OCPQ supports the OCEL 2.0 specification [5] in any of the three introduced exchange formats (JSON, XML, and SQLite). After the dataset was imported, some basic information on the OCEL are shown, including the number of events and objects, as well as all object and event types together with their attributes.

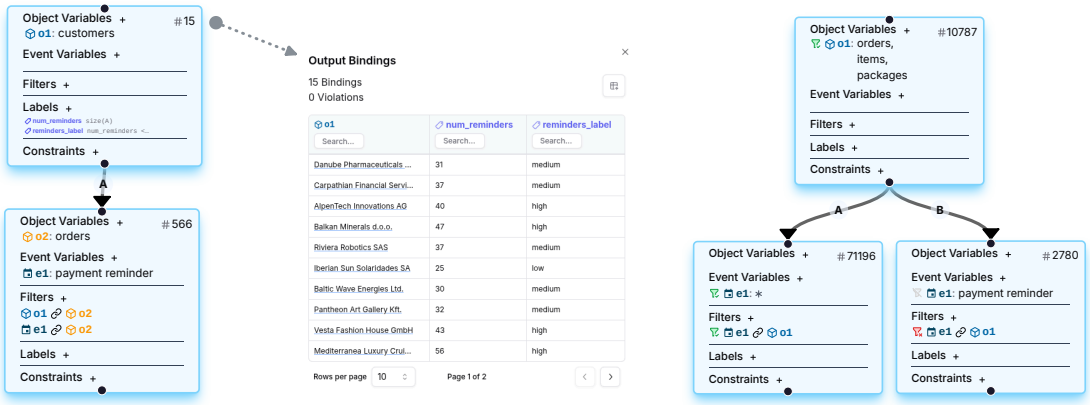

Figure 2: Two advanced features of the OCPQ Tool: On the left, CEL annotations are added, showing the number of payment reminders per customers (`num_reminders`), as well as a category label based on this number (`reminders_label`). On the right, the filter functionality is shown, for example only retaining objects of types `orders`, `items`, and `packages` and their events, and E2O, but excluding E2O for `payment reminder` events.

2.1. Features

After importing an OCEL file, the dataset can be explored using queries, constraints, or graph visualizations of objects, events, and their relationships.

Visual Query and Constraint Tree Editor The query editor appears after creating a new query and can be used to create and link multiple query nodes to form a query tree. This tree structure allows easily modeling and nesting of subqueries (e.g., a subquery for all items in an order). In each node, event and object variables can be added, as well as filter predicates, determining what combinations of object and event instances to consider. For example, the E2O filter predicate specifies that there should be an event-to-object relationship between variable values. Figure 1 shows the user interface of the query tree editor on the left. The top node queries all combinations of `orders` objects and corresponding `place order` events. For that, the node introduces two variables `o1` (for the `orders` object) and `e1` (for the `place order` event), and also has one E2O filter predicate (visualized as a link icon), specifying that the values of `o1` and `e1` should be in an E2O relationship. Queries can be evaluated using the play button on the top right. After evaluation, the number of queried bindings is displayed in the top right for every node, indicated with # as shown in Figure 1. Analogously to filters, predicates can also be used for constraints. For that, bindings that fulfill all constraint predicates are considered satisfied, and violated otherwise. If constraint predicates are used, the violation percentage of a node is used to determine its color. For instance, in Figure 1, the root node is colored yellow because it is violated in around 30% of bindings.

General Annotations and KPIs The violation status is not the only annotation that can be added to output bindings. For example, KPIs, like the total order volume or the number of payment reminders per customer, can be augmented to each output binding row. To allow

general annotations, OCPQ uses the *Common Expression Language*[1]. In Figure 2, an example OCPQ query calculating the number of payment reminders per customer is shown.

All output tables can be explored in the tool directly, or also exported as CSV or XLSX files for usage in other tools and applications, e.g., as input for machine learning techniques.

OCEL Filtering Filtering OCEL datasets and exporting the resulting subset again is also supported in OCQP. The filtering is implemented using three different configuration modes for each element (i.e., object, event, or relation): Included (green) which specifies that the element should be included in the output. Excluded (red) which specifies that the element should be explicitly excluded, even if it is included somewhere else. Ignored (gray) which does not influence the output. On the right of Figure 2, a filtering example of OCPQ is shown.

Other Features Apart from the previously mentioned functionality, OCPQ also has some additional features. For example, it supports automatic discovery for some types of constraints based on an input OCEL. Moreover, some safeguards are in place to stop execution of overloaded queries early on and inform the user. For more information and feature descriptions, we refer interested readers to the website of the tool.

3. Implementation

The OCPQ implementation is based on the Rust4PM software library presented in [8], in particular using its OCEL 2.0 data structures and importers. The backend and frontend of OCPQ are implemented in a modular way, allowing using the tool both as a desktop application[2] and as a hosted web application. After importing an OCEL, it is processed to link object and event references as indicated by their identifiers in relationships. Multiple implementation details support fast execution of queries: First, query execution is parallelized, for example, evaluating subqueries or additional filter predicates in parallel across each considered binding. Second, the ordering and method for binding new object or event variables to values is optimized, reducing the number of unwanted constructed bindings. For instance, when binding a `customers` and a `orders` object with an O2O relationship, not all combinations of all customers and all orders need to be considered. After constructing one binding for each `customers` object, the O2O relationship can be used to directly construct only those binding extensions that also fulfill the O2O filter predicate. While evaluating the query execution performance in detail is outside the scope of this paper, we refer interested readers to the evaluation section in [6]. There, we investigated the runtime of example queries on a real-life dataset with more than one million events and found that every tested example query finished in less than 85ms (0.085 seconds).

4. Conclusion and Future Work

In this paper, we presented the OCPQ tool for querying object-centric process data. As handling object-centric processes requires more flexibility, OCPQ not only allows querying individual

[1] https://cel.dev/

[2] The desktop application uses the tauri framework from https://github.com/tauri-apps/tauri.

events or objects, but any combination of objects and events. Through a graphical constraint editor, nested queries can be modeled in a tree structure without programming experience. The result of queries can be explored inside the tool, and can also be exported. Constraints and other annotations can be added to add additional information and labels to each output binding.

Maturity The first main version of the OCPQ tool was published in September 2024 as v0.5.0. Since then, more than 15 version updates have been published, including additional functionality, like allowing general annotation labels, filtering OCEL files, or macOS support. The tool is stable, and installers for Windows, Linux, and macOS are automatically built for every release. As described in more detail in [6], the tool also supports larger, real-life datasets well.

Future work We want to open the OCPQ tool up for more use cases and backends, for example, executing the modeled queries via SQL or allowing custom extensions in the tool. Moreover, a case study applying OCPQ to a real-life problem would be interesting. For example, using the situation table export functionality to derive custom features for process outcome prediction.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] A. Polyvyanyy, C. Ouyang, A. Barros, W. M. P. van der Aalst, Process querying: Enabling business intelligence through query-based process analytics, DSS 100 (2017) 41–56.

[2] J. M. Pérez-Álvarez, A. C. Díaz, L. Parody, A. M. R. Quintero, M. T. Gómez-López, Process Instance Query Language and the Process Querying Framework, in: Process Querying Methods, Springer, 2022, pp. 85–111.

[3] T. Vogelgesang, J. Ambrosy, D. Becher, R. Seilbeck, J. Geyer-Klingeberg, M. Klenk, Celonis PQL: A Query Language for Process Mining, in: Process Querying Methods, Springer, 2022, pp. 377–408.

[4] S. Esser, D. Fahland, Multi-Dimensional Event Data in Graph Databases, J. Data Semant. 10 (2021) 109–141.

[5] A. Berti, I. Koren, J. N. Adams, G. Park, B. Knopp, N. Graves, M. Rafiei, L. Liß, L. T. genannt Unterberg, Y. Zhang, C. T. Schwanen, M. Pegoraro, W. M. P. van der Aalst, OCEL (object-centric event log) 2.0 specification, CoRR abs/2403.01975 (2024).

[6] A. Küsters, W. M. P. van der Aalst, OCPQ: Object-Centric Process Querying and Constraints, in: RCIS (1), volume 547 of *LNBIP*, Springer, 2025, pp. 383–400.

[7] J. P. Bowen, The Z notation: Whence the cause and whither the course?, in: SETSS, volume 9506 of *LNCS*, Springer, 2014, pp. 103–151.

[8] A. Küsters, W. M. P. van der Aalst, Rust4PM: A Versatile Process Mining Library for When Performance Matters, in: BPM (Demos / Resources Forum), volume 3758 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024, pp. 91–95.