

SOUP: Simplifying Event Knowledge Graph Management for Process Mining

Flavio Corradini¹, Alessio Giacché¹, Sara Pettinari^{2,*}, Barbara Re¹ and Lorenzo Rossi^{1,*}

¹*School of Science and Technology, University of Camerino, Italy*

²*Gran Sasso Science Institute, L'Aquila, Italy*

Abstract

Emerging object-centric process mining techniques exploit event knowledge graphs to analyze event logs from multiple perspectives. However, current approaches rely on general-purpose graph tools that require technical expertise and lack features tailored to process mining. To address these limitations, we introduce SOUP, a user-friendly tool for constructing, navigating, and analyzing event knowledge graphs without manual queries. SOUP automates the creation of event knowledge graphs from event logs, offers filtering and aggregation operations, and supports large-scale data handling. This solution empowers users to conduct object-centric process mining analysis efficiently, facilitating the discovery of executed processes and the identification of related entities.

Keywords

Process Mining, Event Knowledge Graph, Object-Centric Process Mining, Graph Database

Metadata description	Value
Tool name	SOUP
Current version	1.0
Legal code license	MIT
Languages, tools and services used	Python, Angular, Memgraph, Docker
Supported operating environment	Microsoft Windows, GNU/Linux, Mac OS
Download/Demo URL	https://github.com/soup-ocpm/soup/blob/master/README.md
Documentation URL	https://pros.unicam.it/soup
Source code repository	https://zenodo.org/records/17122224
Screencast video	https://pros.unicam.it/wp-content/uploads/2024/06/soup-video-1.mp4

1. Introduction and Motivation

Object-Centric Process Mining (OCPM) has been recognized as an emerging discipline evolving from traditional process mining to address convergence and divergence problems in the analysis of real-life processes [1]. OCPM has been proposed to explore events related to a single case notion, but to different classes of objects [2]. In this context, an Event Knowledge Graph (EKG) is a flexible event data model that captures different aspects of the system behavior, by efficiently

ICPM Doctoral Consortium and Demo Track 2025, October 20-24, 2025, Montevideo, Uruguay

*Corresponding author.

✉ flavio.corradini@unicam.it (F. Corradini); alessio.giacche@studenti.unicam.it (A. Giacché); sara.pettinari@gssi.it (S. Pettinari); barbara.re@unicam.it (B. Re); lorenzo.rossi@unicam.it (L. Rossi)

🆔 0000-0001-6767-2184 (F. Corradini); 0000-0002-5548-9806 (S. Pettinari); 0000-0001-5374-2364 (B. Re); 0000-0002-6872-0616 (L. Rossi)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

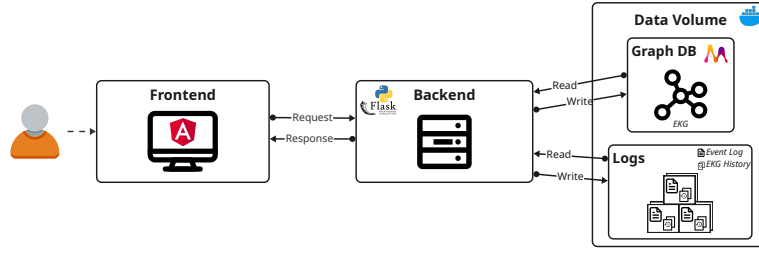


Figure 1: SOUP architecture

storing and querying them using graph database systems [3]. It enables the representation of data recorded in the logs, the correlation between events and objects, and the relations between objects [4, 5], and the inference of the directly-follows relationships from objects’ perspective.

Currently, no tool facilitates the construction, navigation, manipulation, and analysis of EKG from a process mining point of view. Currently, the creation and manipulation of EKGs can be conducted through manual queries [6], which can be intricate and prone to errors, or via libraries [4], that abstract query complexity but rely on graphical interfaces provided by generic graph databases, which are not designed to support and facilitate process mining-based analysis. To fill this gap, the SOUP tool is designed to support researchers in the automatic creation, manipulation, visualization, and analysis of EKGs from event logs, while providing a process mining-oriented interface. SOUP enables users to gain clear and detailed insights into business processes, facilitating the identification of inefficiencies and opportunities for optimization.

The remainder of the paper is organized as follows. Section 2 presents an overview of SOUP, its architecture, and main functionalities. Section 3 explores the impact of the tool. Finally, Section 4 concludes the work and touches upon future extensions.

2. Software Description

The architecture of SOUP, depicted in Figure 1, follows a modular design that separates concerns across frontend, backend, and data storage. The entire application is deployed using Docker containers, with persistent data stored in a dedicated volume.

The **frontend** is developed using the *Angular* (<https://angular.dev>) framework, offering a reactive, single-page application. Graph visualization is supported through the *dagre-d3* (<https://github.com/dagrejs/dagre-d3>) library, which enables dynamic rendering of directed graphs using a process-driven layout. The **backend** is implemented using the *Flask* (<https://flask.palletsprojects.com>) web framework in Python. It serves as a REST API provider for the frontend and handles the interaction with the graph database. The backend is responsible for coordinating data persistence in the data volume and for maintaining analysis histories. The **data volume** acts as persistent storage shared between the application containers. It is logically divided into two key components: (i) *Graph DB*. SOUP uses *Memgraph* (<https://memgraph.com>) as its embedded graph database, hosted inside a dedicated Docker container. Memgraph stores the EKGs and supports efficient querying and manipulation of large graphs. Its native support for the Cypher query language ensures not only efficient data handling but also compatibility with other graph databases, such as Neo4j. (ii) *Logs*. The logs folder stores uploaded event logs and

EventID	Activity	Timestamp	Actor	Order	SupplierOrder	Order_Details	Item	Invoice	Payment	Tray
1	e1	Create_Order	2021-05-01T09:05:00.000	R1	O1		X,X,Y			
2	e2	Create_Order	2021-05-01T09:30:00.000	R1	O2		X,Y			
3	e3	Place_SO	2021-05-01T11:25:00.000	R1		A	X,X,X			
4	e4	Place_SO	2021-05-02T11:35:00.000	R3		B	Y			
5	e5	Create_Invoice	2021-05-03T16:15:00.000	R3	O2			I2		
6	e6	Receive_SO	2021-05-04T10:00:00.000	R2		A	X1,X2,X3			
7	e7	Update_SO	2021-05-04T10:25:00.000	R1	O2	B	Y,Y			
8	e8	Unpack	2021-05-04T10:30:00.000	R2		A	X3			T3
9	e9	Update_Invoice	2021-05-04T10:30:00.000	R2				I2		
10	e10	Unpack	2021-05-04T11:00:00.000	R2		A	X1			T1
11	e11	Unpack	2021-05-04T11:15:00.000	R2		A	X2			T2
12	e18	Create_Invoice	2021-05-05T14:35:00.000	R3	O1			I1		
13	e19	Receive_SO	2021-05-07T10:10:00.000	R2		B	Y1,Y2			
14	e20	Unpack	2021-05-07T10:45:00.000	R2		B	Y1			T2
15	e21	Unpack	2021-05-07T11:00:00.000	R2		B	Y2			T1
16	e27	Pack_Shipment	2021-05-07T17:00:00.000	R4	O1		X1,X2,Y1			T1,T2,T4
17	e28	Ship	2021-05-08T13:00:00.000	R4	O1					
18	e29	Receive_Payment	2021-05-09T08:30:00.000	R5					P1	
19	e30	Clear_Invoice	2021-05-09T08:45:00.000	R5				I1,I2	P1	

Filter Data

Choose the desired fields

Entities	Properties
<input type="checkbox"/> EventID	<input checked="" type="checkbox"/> EventID
<input type="checkbox"/> Activity	<input checked="" type="checkbox"/> Activity
<input type="checkbox"/> Timestamp	<input checked="" type="checkbox"/> Timestamp
<input checked="" type="checkbox"/> Actor	<input checked="" type="checkbox"/> Actor
<input checked="" type="checkbox"/> Order	<input checked="" type="checkbox"/> Order
<input type="checkbox"/> SupplierOrder	<input type="checkbox"/> SupplierOrder
<input type="checkbox"/> Order_Details	<input type="checkbox"/> Order_Details
<input checked="" type="checkbox"/> Item	<input checked="" type="checkbox"/> Item
<input type="checkbox"/> Invoice	<input type="checkbox"/> Invoice
<input type="checkbox"/> Payment	<input type="checkbox"/> Payment
<input type="checkbox"/> Tray	<input type="checkbox"/> Tray

Map the following information

Event Id

Activity Name

Timestamp

Figure 2: SOUP Dataset upload

maintains a history of the analyses performed. This includes JSON-based filter configurations applied during previous sessions. These configuration files can be reused or imported to replicate an analysis without recreating filters manually. For example, a stored filter might include timestamp ranges or specific activities to exclude.

To illustrate the **tool features**, we exploited the small-scale order management event log provided in [7]. The first step involves **uploading data** as a .csv file. SOUP reads and parses this file, displaying a preview to provide users with a visual event table representation. Since this phase results in the creation of an EKG, the user must select which table columns will serve as entities and which as event properties. Each unique value in the selected entity column will create an entity node, and these values will also be saved as event node properties. Moreover, according to the EKG formalization in [7], the user must map the properties column to specify the three essential attributes for building event nodes: the event *identifier*, the *activity* name, and the *timestamp*. Fig. 2 shows the SOUP’s data uploading interface.

Then, the user initiates the **graph-building** process. The backend receives this request and processes the data accordingly. It interacts with the Memgraph database to generate the graph based on the user’s filtered data from the .csv file. Upon successful completion, SOUP provides a summary of the created data in terms of the number of created nodes and relationships and the execution time of each related query. The user can visualize within the card and the pie graph the information about created elements, and the performance of their creation. After graph creation, SOUP offers further operations that the user can perform. The **graph visualization** feature displays the EKG with an initial limit of 200 nodes to ensure smooth rendering performance. Users can dynamically adjust this threshold using a slider to control the number of visualized nodes and their associated relationships. The tool also supports *graph export* functionalities, allowing the user to export the complete graph in either JSON or SVG format. Additionally, graph search functionality allows users to locate specific nodes or relationships within the graph. Finally, the *graph deletion* operation enables users to remove the generated graph from both the Memgraph database and from the logs stored within the data volume.

To facilitate the analysis of EKGs, SOUP provides a range of **filtering** techniques. These include timestamp, frequency, performance, variant, and inclusion/exclusion filters, each designed to refine the graph view and focus on relevant portions of the event data. All applied

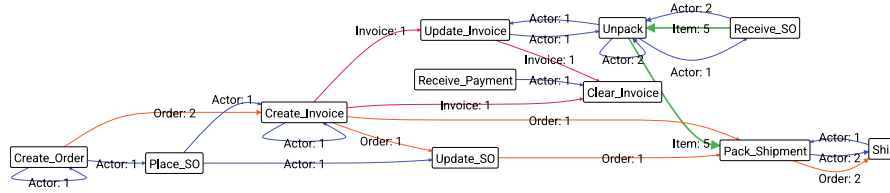


Figure 3: Aggregated graph

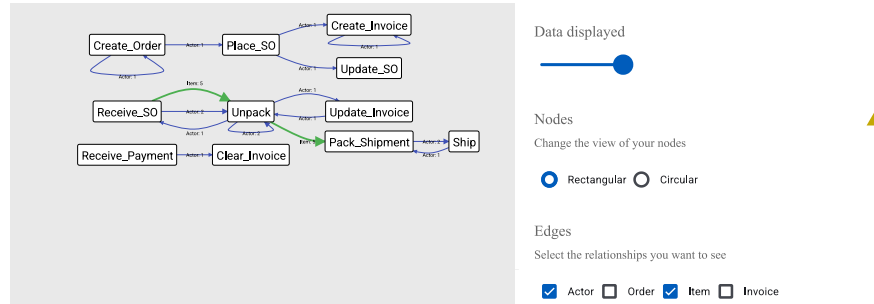


Figure 4: Relationships filtering

filters are stored within the EKG analysis history in the data volume, allowing users to export and re-upload filter configurations for future analyses. The available filters are as follows. *Timestamp Filter* selects all event nodes that fall within a specified time range. *Frequency Filter* is applicable to different entity types. This filter allows users to include or exclude activities based on their frequency using relational operators, i.e., ‘greater than’, ‘less than’, ‘equal to’, or ‘different from’. *Performance Filter* can be applied to filter out traces based on their execution duration, using conditions that use relational operators. *Variant Filter* applies to traces linked to specific entity types. It filters trace variants over an entity type using conditions, expressed via relational operators. Finally, *Inclusion/Exclusion Activity Filter* enables users to select specific activities to include or exclude from the analysis. Filters in SOUP can be composed and are applied in the exact order in which the user adds them. If the application of one filter overrides the criteria of a subsequent filter, SOUP alerts the user.

Finally, the user can use **aggregation functions** to produce an aggregate EKG, also referred to as multi-entity DFG [7]. Specifically, SOUP supports event node aggregation based on the activity name and, optionally, the entity they share. Notably, the tool also indicates, using an alert, whether some event nodes have an entity with a null value, thus suggesting potential data inaccuracies or inconsistencies if the aggregation is performed over that entity type. Based on the user selections, SOUP performs the aggregation query and loads the corresponding graph visualization, see Fig. 3. At this stage, users can leverage various operations, such as adjusting the graph by hiding selected relationship types (Fig. 4), changing the node graphics, or searching nodes inside the graph.

3. Impact and Maturity

The usage of EKGs is rapidly growing within the process mining community, particularly for supporting OCPM analysis. While the OCPA library [8] and the OCEL 2.0 toolsuite [9] provide

solutions for automating the discovery and analysis of object-centric Petri nets and Directed Flow Graphs, there are currently no automated tools designed for EKGs. Existing approaches typically rely on third-party visualization libraries, which are not tailored to the specific needs of process mining. In this context, SOUP introduces several key innovations that fill this gap.

The tool assists users in uploading event logs and identifying relevant events, entities, and properties. This automated approach significantly reduces the time spent on manual data entry and minimizes the potential for errors, compared to traditional query-based methods. Moreover, SOUP offers an intuitive and user-friendly interface for efficient graph investigation and manipulation, thus eliminating the need for advanced database querying skills.

Differently from existing approaches using Neo4j for EKGs, the SOUP tool uses Memgraph as its graph database. This choice has been driven by Memgraph’s improved performance (<https://memgraph.com/blog/memgraph-vs-neo4j-performance-benchmark-comparison>), making it well-suited for managing large numbers of nodes and relationships, common in real-world event logs. Although SOUP is optimized for Memgraph, the tool can be connected to any graph database by adjusting the query language. Moreover, since both Memgraph and Neo4j use the same query language, SOUP can be adapted to include existing solutions designed for Neo4j.

In general, SOUP features an intuitive, dynamic, and easy-to-use design that makes advanced features accessible to all users, regardless of their technical expertise.

The SOUP tool is provided as an open-source Docker container, containing also the Memgraph image. The tool is enriched with example analysis, a demo video, and documentation. It can be easily installed and executed with any event log, as long as they are in a .csv format and contain a timestamp format supported by the Memgraph database. Notably, in this latter case, the tool notifies the user that the uploaded log has an unsupported timestamp format.

SOUP has been assessed with two large datasets, i.e., BPIC17 [10] and BPIC19 [11], that are recognized benchmarks in the process mining community, mainly used for evaluating traditional process mining techniques. In addition, it has been tested on a variety of object-centric scenarios extracted from the literature [7, 12, 9, 13], spanning diverse domains such as order management, robotics, and videogames. For each case study, we recorded the time required to construct the EKG, focusing on the creation of nodes and relationships. These performance metrics are publicly available in the online repository. This evaluation shows the scalability of SOUP, as the tested logs vary in size and originate from heterogeneous application domains, many of which do not natively provide event logs tailored for EKG construction.

4. Conclusion and Future Work

The SOUP tool has been proposed to ease the manipulation and analysis of EKGs. Its development follows the existing EKG formalization [7] and leverages an efficient graph database to handle large event logs. Unlike other approaches that depend on fixed database interfaces or manual query construction, SOUP provides an intuitive, web-based interface that enables users to build and analyze EKGs without requiring specialized technical knowledge. The application of SOUP to several real-world case studies demonstrated the tool’s effectiveness and scalability.

As future work, we plan to increase the flexibility and robustness of the tool. First, we aim to support additional event log formats such as OCEL 2.0, aiming to align SOUP with

all emerging standards in the object-centric process mining community. Second, while the current interface is mainly designed to avoid manual querying, we intend to add support for composing and executing custom Cypher queries, thus providing expert users with deeper analytical capabilities. Moreover, thanks to the versatility of SOUP, it can be easily extended with existing solutions designed for EKGs, such as the PromG library [14], designed to manage and explore object-centric event data and perform OCPM analysis.

Declaration on Generative AI

All research content is original to the authors. No Generative AI has been used.

References

- [1] W. M. P. van der Aalst, Object-Centric Process Mining: Dealing with Divergence and Convergence in Event Data, in: *Software Engineering and Formal Methods*, volume 11724 of *LNCS*, Springer, 2019, pp. 3–25.
- [2] A. Berti, W. M. P. van der Aalst, OC-PM: analyzing object-centric event logs and process models, *Int. J. Softw. Tools Technol. Transf.* 25 (2023) 1–17.
- [3] S. Esser, D. Fahland, Multi-dimensional event data in graph databases, *J. Data Semant* 10 (2021) 109–141.
- [4] A. Swevels, D. Fahland, M. Montali, Implementing Object-Centric Event Data Models in Event Knowledge Graphs, in: *Process Mining Workshops*, volume 513 of *LNBIP*, Springer, 2024, pp. 431–443.
- [5] A. Giacché, S. Pettinari, L. Rossi, Revealing one-to-many event relationships in event knowledge graphs, in: *Process Mining Workshops*, Springer, 2024, pp. 184–196.
- [6] E. L. Klijn, D. Preuss, et al., Event knowledge graphs for auditing: A case study, in: *Process Mining Workshops*, volume 503 of *LNBIP*, Springer, 2023, pp. 84–97.
- [7] D. Fahland, Process mining over multiple behavioral dimensions with event knowledge graphs, in: *Process Mining Handbook*, volume 448 of *LNBIP*, Springer, 2022, pp. 274–319.
- [8] J. N. Adams, G. Park, W. M. P. van der Aalst, ocpa: A python library for object-centric process analysis, *Software Impacts* 14 (2022) 100438.
- [9] I. Koren, J. N. Adams, A. Berti, W. M. P. van der Aalst, OCEL 2.0 resources - www.ocel-standard.org, in: *ICPM Doctoral Consortium/Demo*, volume 3648, CEUR-WS.org, 2023.
- [10] B. van Dongen, BPIC, 2017. doi:10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1.
- [11] B. van Dongen, BPIC, 2019. doi:10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1.
- [12] F. Corradini, S. Pettinari, B. Re, L. Rossi, F. Tiezzi, A methodology for the analysis of robotic systems via process mining, in: *Enterprise Design, Operations, and Computing*, volume 14367 of *LNCS*, Springer, 2023, pp. 117–133.
- [13] L. Liss, N. Elbert, et al., Framework for extracting real-world object-centric event logs from game data, in: *Process Mining Workshops*, Springer, 2024, pp. 363–375.
- [14] A. Swevels, E. L. Klijn, D. Fahland, Object-centric process mining (and more) using a graph-based approach with promg, in: *ICPM Doctoral Consortium/Demo*, volume 3648, CEUR, 2023.