Journal Pre-proof

A Mapping Methodology Enabling Object-Centric Process Mining on Blockchain Applications

Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Barbara Re and Lorenzo Verducci

PII: S2096-7209(25)00119-8
DOI: https://doi.org/10.1016/j.bcra.2025.100392
Reference: BCRA 100392

To appear in: *Blockchain: Research and Applications*

Received date: 18 April 2025
Revised date: 2 July 2025
Accepted date: 23 July 2025

Please cite this article as: F. Corradini, A. Marcelletti, A. Morichetta et al., A Mapping Methodology Enabling Object-Centric Process Mining on Blockchain Applications, *Blockchain: Research and Applications*, 100392, doi: https://doi.org/10.1016/j.bcra.2025.100392.

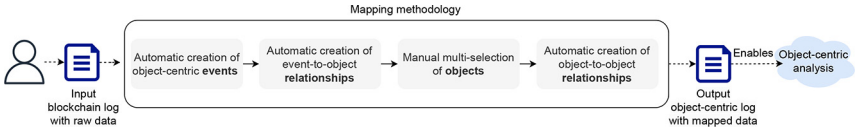
Graphical abstract

Blockchain: Research and Applications ●●●●, ●●●, ●●●

A Mapping Methodology Enabling Object-Centric Process Mining on Blockchain Applications

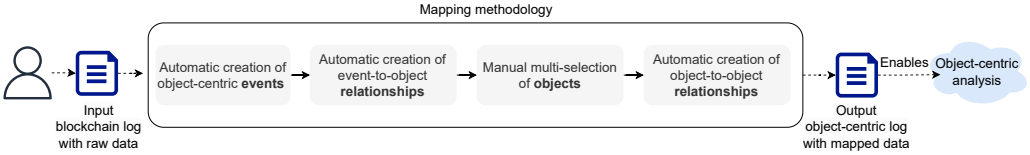
Flavio Corradini , Alessandro Marcelletti ,*, Andrea Morichetta , Barbara Re , Lorenzo Verducci

University of Camerino, Via Madonna delle Carceri 7, Camerino, 62032, Italy



Highlights

- A novel methodology to map blockchain data to object-centric logs.
- The methodology enables the usage of available process mining techniques.
- An open-source web application supporting the proposed methodology was implemented.
- A Real-world project was used to showcase the potential of the methodology.



Graphical Abstract

A Mapping Methodology Enabling Object-Centric Process Mining on Blockchain Applications

Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta,
Barbara Re, Lorenzo Verducci

A Mapping Methodology Enabling Object-Centric Process Mining on Blockchain Applications

Flavio Corradini[a], Alessandro Marcelletti[a,*], Andrea Morichetta[a], Barbara Re[a], Lorenzo Verducci[a]

[a] *University of Camerino, Via Madonna delle Carceri 7, Camerino, 62032, Italy*

Abstract

Blockchain technology has fostered the advent of decentralised applications relying on smart contracts to encode business logic. In this context, it is particularly relevant to analyse data produced by smart contracts that can reveal valuable insights about applications and their execution. In recent years, process mining and, in particular, object-centric techniques have been considered to capture the complexity of blockchain applications and reveal their different perspectives. However, their adoption raises novel challenges related to the object-centric representation of blockchain data. To address this problem, in this work, we propose a mapping methodology guiding users to correlate blockchain data to an object-centric log without requiring advanced technical skills. The proposed methodology generates logs that are compatible with current tools and techniques, enabling in-depth analysis. The methodology was implemented as a web-based tool and its applicability was evaluated on three real-world applications together with a performance and usability assessment.

Keywords: Blockchain, Process mining, Analysis, Decentralised applications, OCEL

1. Introduction

Blockchain enables new forms of trustworthy applications where smart contracts encode distributed interactions and business logic can be executed

*Corresponding author
Email address: alessand.marcelletti@unicam.it (Alessandro Marcelletti)

in an immutable and transparent manner. The execution of smart contracts generates transparent records in the blockchain about several types of data, such as executed activities, involved users, and exchanged information. Thanks to the transparent nature of the blockchain, each recorded operation is accessible and verifiable to everyone, making produced data usable for mining, auditing, and monitoring purposes [1, 2]. Blockchain data can be used as input for analysis techniques to discover valuable insights into the actual behaviour of the blockchain application and its effective execution. Furthermore, such data can uncover unexpected behaviours or vulnerabilities and highlight the architectural and operational logic of the application [3, 4]. In particular, the analysis of the interactions and dependencies between components (e.g., smart contract functions) can improve the structural understanding of practitioners and analysts without the need for in-depth technical knowledge [5].

In this regard, process mining [6, 7] is emerging as a prominent solution for analysing applications starting from data (i.e., event logs) resulting from smart contract executions [8]. It consists of a set of tools and techniques designed to gain insights into processes by identifying bottlenecks, deviations from expected behaviour and opportunities for optimisation [6, 7]. When applied to blockchain execution data, process mining can reveal novel insights that would otherwise remain hidden. To analyse event logs, traditional techniques relies on standardised formats such as XES (eXtensible Event Stream) [9], which require data to be grouped around a unique identifier known as a case. Grouping data by case makes it possible to trace the sequence of events for each process instance under consideration, providing a clear perspective for analysis.

Nevertheless, this single-case perspective does not capture the interactions and relations that occur between multiple events and objects. This limitation leads to several issues that result in misleading statistics or incorrect identification of relations and causalities [10]. Such issues are particularly evident in blockchain analysis, where traditional process mining approaches may prove limited or ineffective due to the complexity and heterogeneous structure of blockchain data [11, 8, 12]. For instance, the execution of a blockchain application involves multiple users interacting with each other and with various smart contracts, resulting in the generation of transactions and emission of events. Analysing such heterogeneous data from a single perspective offers only a partial view of the application's behaviour. Capturing all these dimensions simultaneously thus remains a significant chal-

lenge. For this reason, object-centric process mining [10] has been recently proposed to overcome the limitations of traditional approaches, leading to the development of related standards such as Object-Centric Event Data (OCED) [13] and Object-Centric Event Log (OCEL) [14]. This permits the provision of novel and comprehensive analysis considering all the perspectives of a blockchain application and its execution [15, 16]. The object-oriented structure can represent in a more natural way the contract-oriented design of blockchain applications. Despite this potential, the application of these techniques poses new challenges since blockchain platforms produce data at different granularity levels, like transaction payloads, state updates, and transaction receipts. This granularity requires blockchain data to be properly transformed and converted into an object-centric representation before being effectively used. This preprocessing step is critical [17], and dedicated approaches are necessary. Nevertheless, current solutions in this direction provide support for the generation of custom standard or specific analysis perspectives. Therefore, there is a lack of a comprehensive methodology that enables the analysis of heterogeneous and meaningful information from the blockchain. Moreover, existing approaches do not offer a flexible and user-oriented infrastructure that supports customisable mappings and alleviates the burden of data preprocessing for generating object-centric event logs.

For this reason, in this paper **we propose a mapping methodology to enable object-centric process mining on blockchain applications**. The methodology permits capturing multiple perspectives of an application to discover the executed behaviour and related insights. This is done through a multi-object selection, enabling users to decide which blockchain data to map in an object-centric form in a simple and guided manner. In this way, technicalities are abstracted, making the mapping accessible to a broader audience without requiring advanced technical expertise. The methodology supports Ethereum Virtual Machine (EVM)-based transaction data, including executed functions, events, storage state variables, and internal transactions. We focus on EVM platforms since they are the most used for application development [18, 19]. In this context, executed functions are considered first-class citizens since they represent the application logic, offering a direct view into the behaviour of smart contracts. Finally, as an object-centric standard, we adopt OCEL 2.0 [20], ensuring the produced logs are compatible with existing tools and techniques. The methodology was implemented as a web-based tool and tested on three real-world applications. To evaluate the methodology, we analysed the performance required to generate object-

centric logs and we conducted a usability assessment with the involvement of practitioners.

The rest of the paper is organised as follows. Section 2 introduces the main concepts on which the work relies. Section 3 presents the methodology with its mapping strategies. Section 4 shows the application implementing the proposed methodology. Section 5 evaluates it, showing the methodology applied to different scenarios, and reporting the performance and usability assessments. Related works and comparisons are described in Section 6. Finally, Section 7 concludes the work and discusses future directions.

2. Background

In this section, we describe the concepts forming the foundation of our work. We begin by describing blockchain technology and the data produced by applications that are considered within the proposed methodology. We then proceed with the introduction of object-centric process mining with the OCEL 2.0 standard and its core concepts.

2.1. Blockchain

Ethereum is a decentralised, open-source blockchain with smart contract functionality [21]. Its public and permissionless characteristics permit participants to freely interact with it while ensuring that ledger data is accessible and visible to any interested party. This is possible thanks to the decentralised nature of the blockchain, which, together with cryptographic primitives and consensus mechanisms, ensures accountability and immutability of recorded operations. **Smart contracts** are programs deployed on a blockchain which code is immutable and runs when predetermined conditions are met. They are commonly used to automate agreement processes, ensuring that all participants promptly know the outcome without requiring intermediaries. In Ethereum, these contracts can be coded using the *Solidity* programming language[1], which runs on the Ethereum Virtual Machine, similar to traditional programming languages. Once the code is generated, it is deployed in the blockchain and becomes available for user interaction. Each smart contract has a data area called storage, a persistent key-value store between function calls and transactions. In particular, **state variables** are

[1] `https://solidity.readthedocs.io/`

those variables defined in the smart contract that are permanently stored in the storage over the blockchain. Each executed function of a smart contract can lead to an update of the global state of the ledger, maintained by nodes and containing information about balances and data. Consequently, state variables can be updated over time.

Transactions are cryptographically signed instructions sent on the blockchain by an account. For example, public transactions represent the invocation of a **smart contract function** or the transfer of cryptocurrency. After the execution is completed, the transaction is added to a block and propagated in the network, where its data is included. Among the others, a transaction includes its **hash**, the **block number** where it was included, the **timestamp** of when it was added in a block, the **sender** address creating the transaction, and **gas used**. The latter represents the computational complexity of the execution operation and is used to derive the fee to pay. In addition, a transaction can include an additional log containing **events** emitted during the execution of a smart contract to inform external applications. Those events are customised and can contain many different data, exposing the outcome of an operation (e.g., transfer of a token, deposit).

Internal transactions occur instead between smart contracts and are typically stored off-chain, meaning they are not a part of the blockchain. These can be divided into (I) *call*, executing code of another contract with its context; (II) *staticcall*, like call, but preventing state modifications; and (III) *delegatecall*, executing another contract's code but within the caller's context. When an internal transaction is triggered, the smart contract interacts with the target one to complete the operation. Even in a single transaction, a smart contract can perform several internal calls to other contracts.

While the presented description refers to the Ethereum blockchain, it is worth mentioning that the EVM serves as an execution environment for many blockchain networks. This enables the standardisation between smart contract definition and execution and compatibility across platforms and all applications across EVM-based ecosystems.

2.2. Extracting blockchain data

Extracting meaningful data is a critical step in analysis activities, and various approaches have been proposed. In our previous works [22, 23], we presented an approach and a related tool to extract a comprehensive set of data at different granularities, such as internal transactions and state variables. Once extracted, the approach produces an output log, which serves

as a starting point for further analysis. Given the amount and typology of extracted data, we adopt this approach as the foundation for the mapping methodology proposed in this work. In the following, we describe the extraction approach, focusing on the main features and data we consider in this work.

An important feature of the extraction is the *configuration* permitting the definition of filters and parameters to delimit the transactions to extract. This information defines:

- *Network.* The target EVM-based network
- *Block range.* Range of blocks from which transactions are extracted
- *Contract address.* Address of the deployed smart contract
- *Contract name.* Name of the target smart contract
- *Gas used range.* Range of units of gas used filtering transactions to extract
- *Gas price range.* Range of gas prices filtering transactions to extract
- *Interval of time.* Interval of execution timestamps used to select transactions for extraction
- *Set of sender addresses.* List of addresses to consider for transaction senders
- *Set of executed functions.* Names of the functions executed considered for the transactions to extract

Once filters are configured, matching transactions are extracted and decoded together with other relevant pieces of data. Figure 1 depicts the conceptual model of the log, specifically on the data we extract and its relationships, based on the blockchain concepts previously introduced. Listing 1 shows instead the structure and technicalities of the resulting log, having the **transaction** as a base element, representing the execution of a smart contract function (Line 2). In the log, a transaction is composed of a hash (Line 3) and the block number (Line 4) identifying the block in which it is included. The contract address (Line 5) is the address of the smart contract to which the transaction is directed, while the sender (Line 6) identifies the address

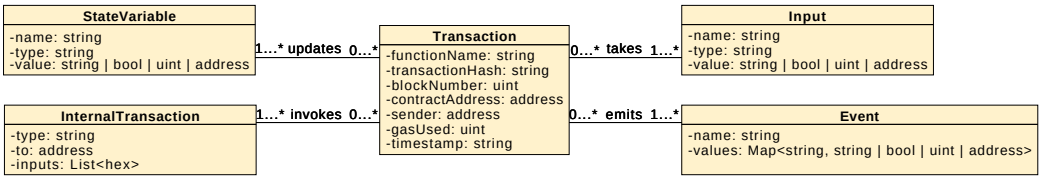


Figure 1: Conceptual model of the considered blockchain log

of the blockchain account sending the transaction. Gas used (Line 7) is the amount of gas used for executing the transaction. Finally, the timestamp (Line 8) is when the transaction was sent, and the format used is ISO8601. The remaining part of the log contains structured and detailed information about the executed function.

Listing 1: Blockchain Log

```
1  {
2      'functionName': "anActivity",
3      'transactionHash': "0xc4f28...",
4      'blockNumber': 103450,
5      'contractAddress': "0xa3d61...",
6      'sender': "0xcb3d9...",
7      'gasUsed': 100000,
8      'timestamp': "2023-12-02T10:05:23.000Z",
9      'inputs': [...],
10     'storageState': [...],
11     'events': [...],
12     'internalTxs': [...]
13 }
```

When a function is executed, it could take one or more **input** parameters, which example is shown in Listing 2. Inputs have a name (Line 16), a type (Line 17) and a value (Line 18).

Listing 2: Inputs array

```
14 'inputs': [
15   {
16     'inputName': "anInput",
17     'type': "address",
18     'inputValue': "0xe5d37..."
19   },
20   ...
21 ]
```

The execution of a smart contract updates its state, specifically, its storage state variables, which are variables defined inside the smart contract and which persist in the blockchain. An example is shown in Listing 3. As standard variables, a state variable has a name (Line 24) and a type (Line 25) corresponding to a Solidity type (e.g., unit, string, bool) and a value (Line 26) that can be modified upon execution of the contract. Since the value is decoded through the proposed approach, additionally, the original value in hexadecimal format is included (Line 27).

Listing 3: Storage state

```
22 'storageState': [
23   {
24     'variableName': "variableName",
25     'type': "t_uint16",
26     'variableValue': 0,
27     'variableRawValue': "0000"
28   },
29   ...
30 ]
```

Inside a transaction, an additional log can be included named **event** which example is shown in Listing 4. An event is custom data specified within the smart contract and used to inform external entities about a certain occurring event. A blockchain event has a name (Line 33) and can include multiple topics associated with it and each of them has a name and a value (Lines 35 - 38). To avoid confusion with the OCEL 2.0 event, henceforth we refer to this kind of data as *blockchain event*.

Listing 4: Blockchain events

```
31 'events': [
32   {
33     'eventName': "anEvent",
34     'eventValues': {
35       'eventTopic1': "0x39739...",
36       'eventTopic2': "1234567...",
37       ...
38       'eventTopicN': "...",
39     }
40   },
41   ...
42 ]
```

A smart contract can call another contract to execute a certain operation producing an **internal transaction** which example is reported in Listing 5. These interactions can be of different types (Line 45), are directed to a target contract address (Line 46) and can have some input parameters (Lines 47 - 49).

Listing 5: Internal transactions array

```
43 'internalTxs': [
44   {
45     'callType': "CALL" || "STATICCALL" || "DELEGATECALL",
46     'to': "0000000000000000000000004d73adb...",
47     'inputsCall': [
48       "4d3a0f7c000000000000000000000000152...",
49       ... ]
50   },
51   ...
52 ]
```

2.3. Object-centric Process Mining

Process mining can be used to identify deviations from the expected behaviour of the monitored application, but also to improve underlying processes [6, 7]. The starting point for this kind of analysis is event logs that contain event data representing the various executed activities. Several formats have been proposed to standardise the representation of event data, with the eXtensible Event Stream (XES) [9] being the most widely used so far. Each event requires a single case identifier (referring to the instance in which the event occurred), a timestamp (when the event occurred), and an activity label describing its type. Events can also have additional attributes included in the log. The concept of a *case* represents a unique instance of a process, such as a customer order, where all related events are grouped together under one identifier. This grouping captures the sequence of activities associated with that specific case, enabling the analysis of the process flow from initiation to completion. However, this case-centric approach assumes there is just one case identifier and that each event refers to precisely one. Consequently, it presents limitations in real-world scenarios where processes involve multiple interacting entities. In a blockchain setting, these issues are consequently reflected, since an application contains multiple and fine-grained data connected to each other.

Consider a simple blockchain transfer function that involves both a sender and a receiver. This single function is logically connected to two distinct entities, the sender's account (*from* data field) and the receiver's account (*to* data field). If the sender's account is selected as the case identifier, one can observe all transfers initiated by that account. However, the transfers received by the corresponding receiver remain hidden. On the other hand, if the receiver's account is chosen as the case identifier, the transfers initiated by the sender are not visible. Therefore, by selecting only one entity as the case identifier, the comprehensive view of the interaction between sender and receiver is lost. When considering more complex applications, involving multiple events, contract-to-contract interactions, and the evolving state, these limitations are even more significant.

To overcome such issues, object-centric paradigms were proposed, allowing the observation of systems from multiple perspectives and considering the variety of objects involved during the execution of activities [10]. Considering the previously mentioned transfer scenario, an object-centric approach addresses this limitation by representing both the sender and the receiver as distinct objects associated with the same transfer function. Instead of forcing the analysis to follow a single case notion, each event is linked to all relevant objects involved, allowing for a comprehensive representation of the process. This enables, for instance, the analysis of the interactions between accounts over time.

Object-Centric Event Log. To practically support object-centric analysis, some format and standards were proposed. Among the available ones, OCEL 2.0 is one of the most established to define and operate with object-centric event logs. It forms the basis for object-centric process mining by enabling the expression of events, objects and their relationships. OCEL 2.0 is also supported by existing object-centric process mining techniques, making it a valid solution for this kind of analysis. OCEL 2.0 comes with a metamodel shown in Figure 2 representing its main concepts and components. At the basis, we find **Events**, representing the execution of certain activities within a system. Events are unique, typed and associated with a timestamp. **Event types** categorise the different events depending on their purpose, representing specific actions. An event refers exactly to one event type. Other core elements are **Objects**, expressing the entities involved during the events. Objects can refer to physical or abstract concepts and have different attributes with values changing over time. **Object types** categorise the different kinds of

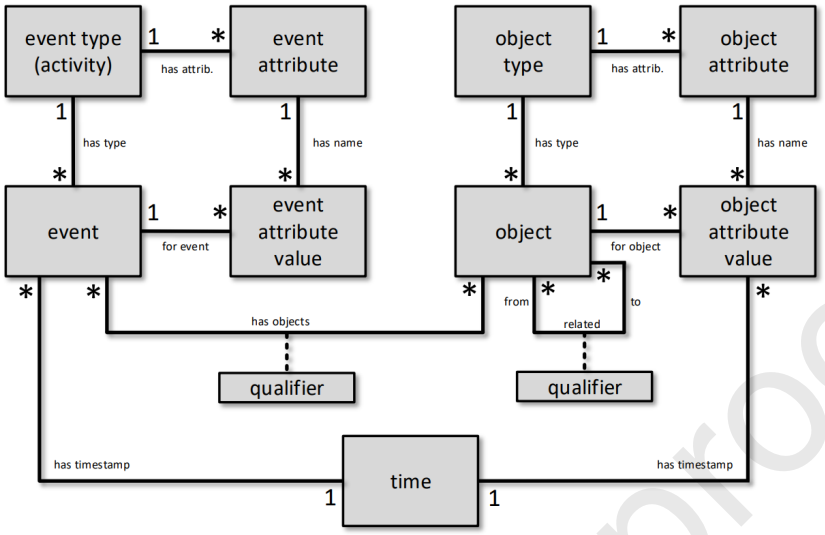


Figure 2: OCEL 2.0 metamodel from [20]

objects present in the system.

In OCEL 2.0, presented entities can be associated according to some kinds of relationships. **Event-to-Object (E2O) Relationship** describes the relationship occurring between events and objects. This relationship can be qualified, by describing the roles that objects have in specific events. **Object-to-Object (O2O) Relationships** represent the case where objects are related to other objects. Also in this case, the relationship can be qualified.

3. Mapping Methodology

To enable the adoption of object-centric process mining for blockchain-based applications, a crucial phase is the preprocessing of blockchain data to make it suitable for analysis. For this reason, we propose a mapping methodology designed to support analysts and domain experts in generating usable object-centric logs from the fine-grained data recorded on the blockchain. The methodology consists of some phases for associating blockchain data with object-centric elements. A key feature is the multi-object selection, which allows users to decide how to generate logs according to different kinds of blockchain data and consequent mapping strategies.

The design of our mapping methodology is motivated by the structure

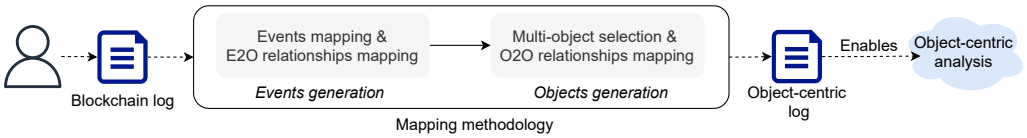


Figure 3: Mapping methodology and its phases

of blockchain applications. Since the execution of smart contract functions corresponds to application-level activities, we treat these as first-class citizens. This is done by working with transaction data, which records executed functions and forms the foundation for constructing OCEL 2.0 logs. An advantage of this design is its general applicability. Transactions represent the execution of a function, regardless of the smart contract typology or the application domain. A further discussion on the generalisability of the methodology can be found in Section 5.

The methodology is depicted in Figure 3 and encompasses two main phases: *events generation*, and *objects generation*. The first phase generates the OCEL 2.0 events and builds the E2O relationships by performing an automatic mapping. The second phase generates the OCEL 2.0 objects and O2O relationships. Differently from the events, the object mapping is made by the user who can select the elements of the blockchain that will be mapped to objects. In the following, we detail the various phases of the methodology. Notice, given the lack of a common standard for representing blockchain data, the mapping approach behind the proposed methodology is based on the blockchain concepts introduced in Figure 1, while the practical implementation relies on Listing 1. However, the core mapping idea can be applied to differently structured data.

3.1. Events generation

The first phase of the methodology corresponds to the generation of the OCEL 2.0 events. This is done by automatically mapping executed smart contract functions to object-centric events. An event represents the execution of an activity in a system, with an event type indicating the type of activity. Similarly, a transaction represents the execution of an activity within the blockchain (i.e., the execution of a smart contract function). For this reason, each executed function is mapped to an OCEL 2.0 event. E2O relationships are also automatically mapped according to a predefined strategy.

Listing 6 shows the mapping of extracted smart contract functions with

events according to the OCEL 2.0 specification. First, the **event type** is defined with a name (Line 55) corresponding to the invoked smart contract function. As attributes (Lines 56 - 68) other relevant fields of the transaction corresponding to the function are used. The event type contains the gas used (Line 58), the block number where the transaction was included (Line 62) and the sender (Line 66). These attributes are useful to detail the transaction, representing its computational cost, the source block and who interacted with the smart contract. The types of the event attributes (Lines 59, 63, and 67) are integer and string. The second part of Listing 6 shows the generation of the OCEL 2.0 *event*. Each created event is identified by an ID (line 72) that we associate with the transaction hash, as it is unique in the blockchain. As type (Line 73) an event must refer to one of the previously defined types. Each event has a time (Line 74) corresponding to the transaction's timestamp converted to the ISO 8601 format. The event attributes (Lines 75 - 88) are then created with the gas value used for the transaction and the sender's address. Finally, the relationships (Lines 89 - 94) with the OCEL 2.0 objects are generated.

Listing 6: OCEL 2.0 event type

```
53     'eventTypes': [
54       {
55         'name': <functionName>,
56         'attributes': [
57             {
58                 'name': "gasUsed",
59                 'type': "integer"
60             },
61             {
62                 'name': "blockNumber",
63                 'type': "integer"
64             },
65             {
66                 'name': "sender",
67                 'type': "string"
68             }
69         ]}...],
70   'events': [
71         {
72             'id': <transactionHash>,
73             'type': <functionName>,
74             'time': <timestamp>,
75             'attributes': [
```

```
76                  {
77                      'name': "gasUsed",
78                      'value': <gasUsed>
79                  },
80                  {
81                      'name': "blockNumber",
82                      'value': <blockNumber>
83                  },
84                  {
85                      'name': "sender",
86                      'value': <sender>
87                  }
88              ],
89              'relationships': [
90                  {
91                      'objectId': <anObjectId>,
92                      'qualifier': <aQualifier>
93                  }
94          ]}...]
```

These relationships serve to associate events with objects, each of them requires an object id (Line 91) and a qualifier (Line 92) that varies depending on the type of the target object. In the proposed methodology, relationships are automatically created. Indeed, we define a set of predefined and admittable relationships and corresponding qualifiers which are shown in Table 1. These relationships are derived from the behaviour of objects within the blockchain context. This is possible since data is consistently structured inside the blockchain. For example, transactions always include a sender, allowing the relationships to be standardised effectively. In this way, we alleviate the burden of manual configuration for the user and the need for domain knowledge.

3.2. Objects generation

The second phase of the methodology consists of objects generation. Objects are crucial elements in an object-centric analysis as they represent the components involved during the execution of a smart contract and express the perspectives of the application. This phase includes a multi-object selection where the user decides which blockchain data should be mapped as an object and included in the final log. Depending on the selected blockchain data, the OCEL 2.0 object is built according to different mapping strategies. As for the E2O case, the O2O relationships are built automatically during

Table 1: Event-to-object relationship qualifiers

Event	**Qualifier**	**Object type**
Function	Generated by	Contract address
	Hash	Transaction hash
	Sent by	Sender
	Takes	Input
	Updates	State variable
	Emits	(blockchain) Event
	Invokes	Internal transaction

this phase. In the proposed methodology, an object can be selected among seven different blockchain data types: *contract address*, *transaction hash*, *sender*, *input*, *state variable*, *blockchain event*, and *internal transaction*.

In mapping blockchain data to OCEL 2.0 objects, we adopted a design that reflects both the peculiarities and the meaning of the involved components. *Contract address* and *sender* are mapped as objects based on their unique values, ensuring that each distinct entity is modelled only once. Indeed, a certain value always uniquely identifies the same smart contract or user over different transactions. In contrast, *inputs* and *events* could have a different meaning within each transaction context, and thus they always generate new objects. For instance, despite a blockchain event *transfer* can be seen as always the same type, it can represent every time an operation between different senders and receivers. The same design was adopted for *internal transactions*, which generates different objects to provide a structured perspective on contract-to-contract interactions, highlighting the roles that various contracts play within a given process. Finally, a different design was adopted for *state variables*, which encapsulate the dynamic behaviour of smart contracts. These are mapped as evolving objects whose attributes are updated over time to reflect changes in the contract's state. This means that a state variable *account* generates only one object, but has an attribute value constantly updated.

Listing 7 presents all the mapping strategies for OCEL 2.0 objects in a compact format. Specifically, it shows, for each OCEL 2.0 property, the possible selections available to the user. A complete and detailed version is

accessible in the technical report online[2]. The mapping begins by defining the various object types, which represent the different kinds of objects. An object type is identified by a name (Lines 96 - 97) and it can be selected from a set of predefined options (shortened for the sake of readability). Some object types are predefined and static (e.g., txHash, sender), meaning their names remain constant regardless of the specific log content. Others are generated dynamically based on actual values found in the log (e.g., varName, eventName). For example, the object type for the sender is always labelled as *"sender"*. In contrast, object types derived from state variables are named based on the variable's name in the log, such as *"transfer"* or *"approve"*. Each object type also includes a set of attributes (Lines 98 - 105), which names (Lines 99 - 101) are automatically generated depending on the selection. Similarly, the corresponding attribute types are set (Lines 102 – 104). Once the object types are defined, the specific objects are created. Each object is identified by an ID (Line 109 - 111), which is automatically composed based on the selected type. The object also includes the type (Lines 112 - 113) and the set of actual attribute values(Lines 119 - 121). Attributes are further characterised by a time property (Line 118). As with events, each object includes a time corresponding to the timestamp of the transaction, converted into the ISO 8601 format.

Listing 7: OCEL 2.0 object mapping strategies

```
95  'objectTypes': [{
96    'name': "cAddr" || "txHash" || "sender" || <inputName> ||
97            <varName> || <eventName> || "internalTx",
98     'attributes': [{
99       'name': "caddr" || "hashValue" || "senderAddr" ||
100              "inputValue" || "varValue" || <eventTopic> ||
101              "Type" || "to",
102      'type': "string" || "string" || "string" ||
103              <typeofInputValue> || <typeofvarValue> ||
104              <typeofEventTopic> || "string" || "string"
105    }],
106 }],
107
108 'objects': [{
109   'id': <cAddr>||<txHash>||<sender>||"input_<i>_<txHash>" ||
110         "var_<varName>_<caddr>" || "event_<i>_<txHash>" ||
```

[2]https://pros.unicam.it/wp-content/uploads/2025/07/TechReport.pdf

```
111         "internalTx_<i>_<txHash>",
112   'type': "caddr" || "txHash" || "sender" || <inputName> ||
113           <varName> || <eventName> || <callType>,
114   'attributes': [{
115     'name': "caddrValue" || "hashValue" || "senderAddr" ||
116             "inputValue" || "varValue" || <eventTopic> ||
117             "callType" || "to",
118    'time': <timestamp>,
119    'value': <caddr> || <txHash> || <sender> || <inputValue>
120             || <varValue> || <eventTopic> || <callType>
121             || <to>
122      }],
123    'relationships': [{
124        'objectId': <anObjectId>,
125        'qualifier': <aQualifier>
126      }, ...]
127 }, ... ]
```

After the attributes, the object contains the O2O relationship (Lines 123 - 126), expressing the relationship that the object has with another one. The relationship is defined by the id of the target object (Line 124) and the qualifier characterising it (Line 125). While the E2O relationship could be standardised by the connections that data have within the blockchain, the O2O strongly depends on the domain knowledge of the application and the low-level smart contract code. Furthermore, objects can be connected with objects in any event, making the definition of default relationships even more complex. For this reason, we have standardised relationships and qualifiers only for some of the available objects as shown in Table 2. Furthermore, relationships are applied only between objects on the same event, so in the same transaction. As for the E2O case, these qualifiers are derived from the blockchain domain depending on how objects interact. For example, a state variable is always part of the storage of a smart contract, permitting the standardisation of their relationship. On the contrary, the update of a state variable may not depend on the input of the executed function or on the user sending the transaction, thus limiting the relationship creation.

4. Implemented Methodology

To enable the practical usage of the methodology, we implemented it and made it available through a web application. Specifically, the methodology was integrated into the web application supporting the data extraction

Table 2: Object-to-object relationship qualifiers

Object type	**Qualifier**	**Object type**
Contract address	Invoked by	Sender
	Contains	State variable
	Defines	Event
Sender	Invokes	Contract address
	Passes	Input
	Generates	Transaction hash
Transaction hash	Triggers	Internal transaction
	Created By	Sender
Input	Inserted by	Sender
State variable	Storage of	Contract address
Blockchain event	Member of	Contract address
Internal transaction	Triggered in	Transaction hash

proposed in [22, 23]. This work extends that approach by incorporating a mapping phase with dedicated functionalities for multi-object selection and log generation. Figure 4 depicts the application components with a highlight on the implemented mapping capabilities. The tool is composed of three main artefacts: *extraction*, *query* and *mapping*. The frontend provides a graphical interface for accessing the respective pages of each component, which are linked to corresponding backend services responsible for executing the required logic. For the *extraction* module, the service interacts with the blockchain and smart contracts to retrieve relevant data. In the case of the *query* module, it connects to the database, where transactions are stored upon extraction. In the following, we provide an overview of the existing extraction and query functionalities, focusing then on the mapping capabilities. The implemented application is openly accessible at `pros.unicam.it/blockchain-mapping/`.

4.1. Extraction and Query Pages

Figure 5 shows the interface for the *Data Extraction* phase, which contains the functionalities implementing the extraction presented in Section 2.2. The page offers several configuration options described in Sec. 2 to enable transaction extraction. *Filters* (I) define parameters for selecting relevant transactions, while *Network selection* (II) allows extraction from different blockchain networks. *Contract configuration* (III) specifies the target smart

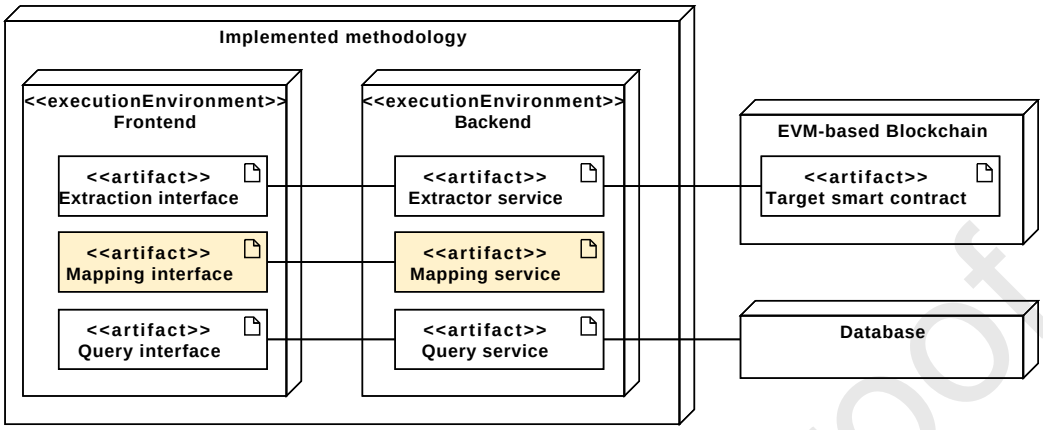


Figure 4: Deployment diagram of the implemented methodology

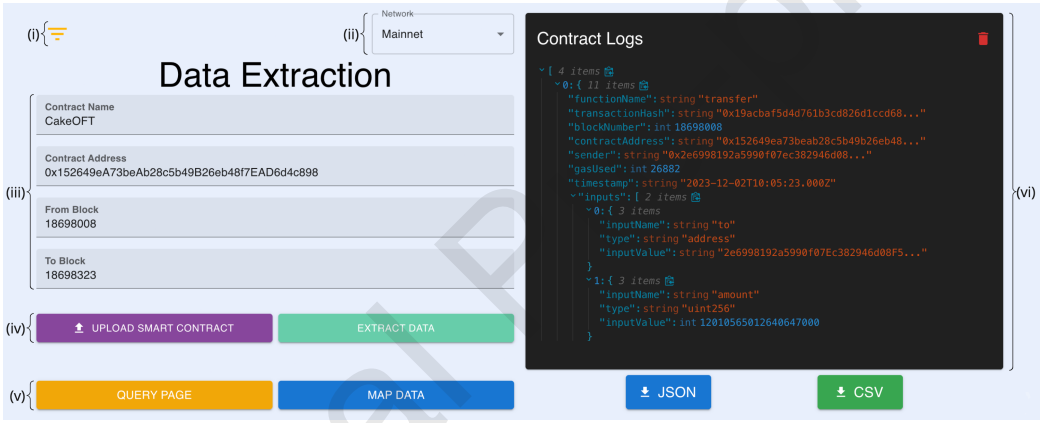


Figure 5: Extraction Page

contract. The extraction process involves two steps with dedicated functionalities (IV). *Uploading the smart contract* to compile it and retrieve necessary data (if not verified, users can manually upload the Solidity source code) and *Extract Data* to initiate extraction and decoding. Additional features include (V) the *Query Page* for database queries, *Map Data* for generating object-centric logs. Finally, the *Extraction outcome* panel (VI), which displays extracted logs and allows downloads in JSON or CSV format.

During the extraction, data is saved in a local database, accessible by the user with querying capabilities. This enables faster data retrieval while the usage of a standard DBMS permits the definition of complex queries and aggregation features. These capabilities are accessible in the query page

shown in Figure 6

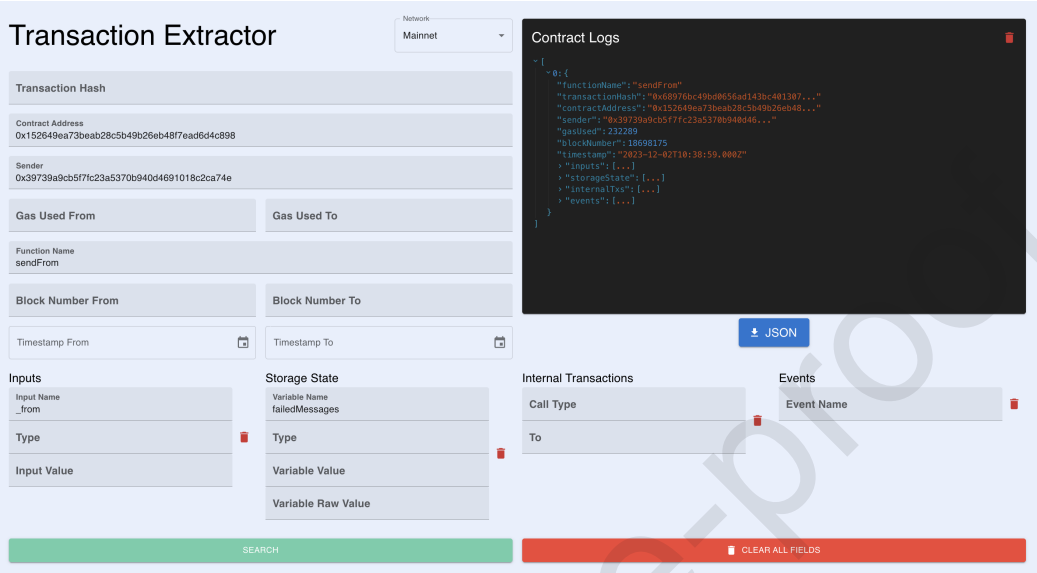


Figure 6: Page for querying transactions from local database.

Here, the user can specify information about a single transaction or a set of transactions by selecting various fields. These fields include transaction hash, contract address, sender, gas used range, function name, block numbers, and timestamps. Additionally, users can apply additional fields related to the log objects previously described (i.e., inputs, storage state, events and internal transactions) to search for particular values or types of data. Once the query is completed, all corresponding transactions are displayed in the right-side panel and can be downloaded as a JSON file.

4.2. Mapping Page

The mapping page permits the creation of the OCEL 2.0 event log starting from the previously extracted blockchain data. Figure 7 depicts the automatic mapping of events, which maps the function executed by the transactions with the name of the event types in the OCEL 2.0 standard, associating the *gas used*, the *block number* and the *sender* attributes.

Figure 8 shows instead the multi-object selection. Here, the user can choose among the various fields in the blockchain log and map them as object types in the OCEL 2.0 standard according to the mapping strategies described in Sec. 3. Through the interface, the user can add multiple object

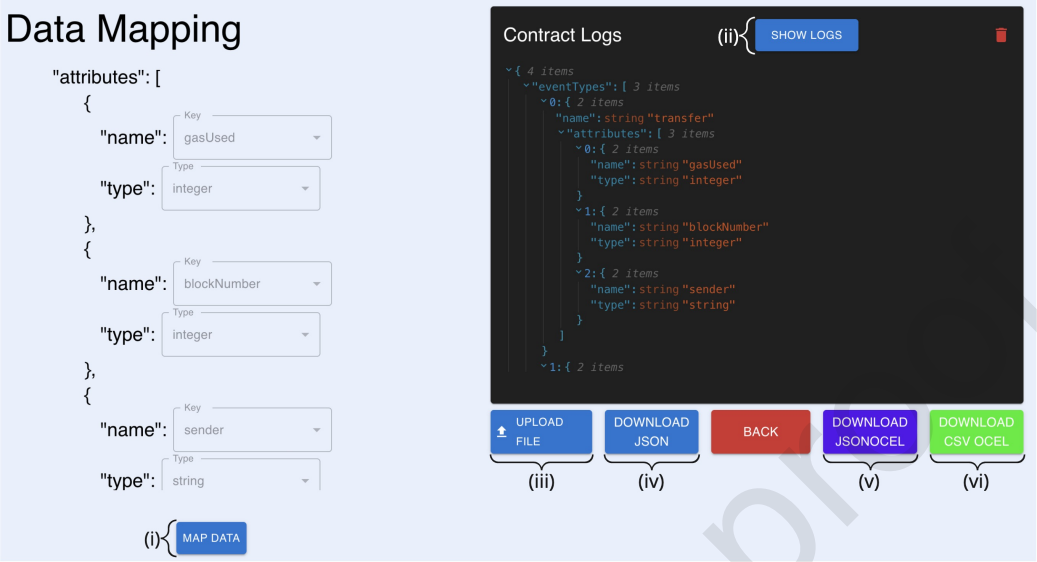


Figure 7: Automatic mapping of event types

types and click on them to display a list with the supported blockchain entries for the mapping. After the selection of objects, clicking the *map data* button, the application generates the OCEL 2.0 log, showing it in the right-side panel.

Finally, additional functionalities are available with dedicated buttons depicted in Figure 8:

I *Map Data*: used to confirm the mapping of the selected object types;

II *Show Logs*: used to change the view between the blockchain log and the OCEL 2.0 one;

III *Upload File*: used to upload a *.json* file with the blockchain log to map;

IV *Download JSON*: permits the download of the OCEL 2.0 log in a JSON format;

V *Download JSONOCEL*: permits to download the OCEL 2.0 log in a *.jsonocel* file;

VI *Download CSV OCEL*: permits to download the OCEL 2.0 log in a flattened CSV format.

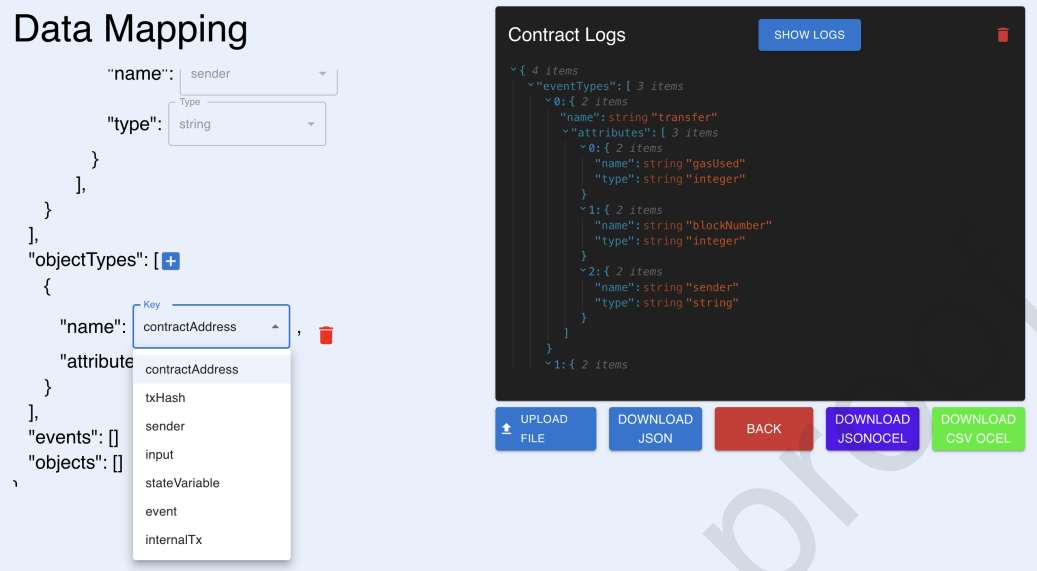


Figure 8: Manual selection and mapping of object types

5. Evaluation

This section evaluates the practical feasibility of the proposed methodology by demonstrating its real-world applicability, analysing its performance, and assessing its usability.

5.1. Usage on Real-world Applications

As a first step, we demonstrate the usage of the implemented methodology on three real-world blockchain-based applications. In particular, we showcase the effectiveness of the mapping by extracting data from the PancakeSwap, CryptoKitties, and RotoHive applications, generating OCEL 2.0 logs, and using them to uncover insights into the executed smart contracts. It is worth mentioning that the main objective is to demonstrate that the methodology can be effective in generating usable and meaningful logs. Indeed, it is not our intention to analyse in detail the mentioned applications, but rather to show some potential insights that should be further explored by interested auditors. The resulting logs are accessible online[3].

[3] https://pros.unicam.it/blockchain-mapping/

PancakeSwap. The first application we consider is PancakeSwap, a decentralised exchange on the BNB Smart Chain that operates across multiple platforms due to its cross-chain capabilities. PancakeSwap enables users to swap BEP20 tokens without the need for middleman services. The CakeOFT smart contract, considered in this context, is used for cross-chain bridging of CAKE tokens on PancakeSwap, and the extracted functions include those responsible for handling the tokens. With CakeOFT it is possible to send tokens from the sender's address to another specified address using *transfer*, an inherited function from the ERC20 protocol; further, CakeOFT allows the transfer of the ownership of this contract to a new account (function invokable only by the current owner of the contract) through the *transferOwnership* function. Other features provided by CakeOFT are: the sending of CAKE tokens from one chain to another using the *sendFrom* function, to allow a third party to transfer tokens from one account to another, up to a certain amount, on behalf of the owner with *transferFrom*, and grants a spender permission to transfer tokens from the token holder's account, up to a specified amount, through *approve*, of ERC20.

To evaluate the methodology, we consider an extensive sample of transactions, analysing the produced object-centric log and its elements. In particular, we extracted transactions from one of the PancakeSwap smart contracts, selecting all possible objects during the mapping phase. The objective is to capture all possible perspectives of the application to discover and reveal insights about created elements, relationships and more. To this purpose, the generated log was used in the Ocelot[4] and OCPM[5] tools to discover the output process, showing related information about events, objects, and relationships. These tools represent current solutions to analyse object-centric logs, providing graphical and statistical insights into the considered application [24].

In the first step, we extracted 1000 transactions from block 16970567 to 17125165 on the Ethereum version of the CakeOFT smart contract[6] without any particular configuration. An example of the extraction output is shown in Listing 8, and it contains the function *sendFrom* updating a state variable *feeOwner*.

[4] `https://ocelot.pm/`
[5] `https://www.ocpm.info/`
[6] `https://etherscan.io/address/0x152649ea73beab28c5b49b26eb48f7ead6d4c898`

Listing 8: Example of log entry for a single transaction.

```
'functionName': "sendFrom",
'transactionHash':"0x18c2d7c7...49b142a6f5",
'blockNumber': 16976687,
'contractAddress':"0x152649ea...ead6d4c898",
'sender':"0x88274e4c...de3266e615",
'gasUsed': 219923,
'timestamp': "2023-04-04T16:25:47.000Z",
'inputs': [
    1: { 'inputName':"_dstChainId",
         'type':"uint16",
         'inputValue': 102 } ... ],
'storageState': [
    0:{ 'variableName': "feeOwner",
        'type': "t_address",
        'variableValue': "0x702099e2...a742782fc2",
        'variableRawValue': "0000000000...a742782fc2" } ...
          ],
'internalTxs': [
    0: { 'callType': "CALL",
         'to': "0000000000...8401a178e2",
         'inputsCall': [
            0: 4d3a0f7c00...26eb48f7ea, ... ]} ... ],
'events': [
    0: { 'eventName': "SendToChain",
         'eventValues': {
             '_dstChainId': 102,
             '_from': "0x88274e4c...de3266e615",
             '_toAddress': "0x0000000000...e3266e615",
             '_amount': 2.79e+21 } ... ]
```

Once extracted, the blockchain log is mapped to the object-centric one according to the presented strategy. To have a complete overview of all possibilities, we selected each case from the blockchain log as an object type. An excerpt of the resulting OCEL 2.0 log is shown in Listing 9. This example shows an event corresponding to the executed function *sendFrom* and an object for the *feeOwner* state variable, having an E2O relationship *updates*. An additional *storage of* O2O relationship is also created between the *feeOwner* and *contract address* objects.

Listing 9: CakeOFT event types

```
'eventTypes': [
    { 'name': "sendFrom",
      'attributes': [
```

```
            { 'name': "gasUsed",
              'type': "integer" },
    ... ]},
      ...,
],
'events': [
    ...
    { 'id': "0x18c2d7c7...49b142a6f5",
      'type': "sendFrom",
      'time': "2023-04-04T16:25:47.000Z",
      'attributes': [
        { 'name': "gasUsed",
          'value': 219923 },
        ...],
      'relationships': [
        { 'objectId': "variable_feeOwner_0x152649ea73...
           ead6d4c898",
          'qualifier': "updates" },
        ...]},
    ...
],
'objectTypes':
[{
      'name': "feeOwner",
      'attributes': [
        {
          'name': "variableValue",
          'type': "string"
        },
        ...
      ]
    }, ... ],
'objects':
[{
      'id': "variable_feeOwner_0x152649ea73...ead6d4c898",
      'type': "feeOwner",
      'attributes': [
        {
          'name': "variableValue",
          'time': "2023-04-04T16:25:47.000Z",
          'value': "0x702099e284...a742782fc2"
        }
      ],
      'relationships': [
        {
```

```
            'objectId': "0x152649ea73...ead6d4c898",
            'qualifier': "storage of"
        }, ... ]
    }, ...
    ]
```

After mapping the elements, we used the OCPM application with the OCEL 2.0 log to visualise the process schema underlying the considered smart contract. Specifically, the OCPM application discovers the object-centric Directly-Follows Graph (DFG) of the executed smart contract. The result is depicted in Figure 9 and shows the extracted functions with the connected objects. Other insights about specific occurrences and relationships are instead obtained from the Ocelot application.

Analysing the obtained log we can find 5 different event types corresponding to the kinds of executed functions. In total, 1000 events are created, one for the extracted transaction, as each refers to exactly one function. A detailed view of all created events is visible in Figure 10. A notable aspect is the number of *approve* events, occurring 825 times. This high difference from other events depends on the PancakeSwap logic, which requires users to approve the trading operations so that the smart contract can effectively transfer the tokens on behalf of the user. Since token trading is the most common operation on a decentralised exchange, the approval operation is consequently the most frequently executed function.

Considering the objects, 29 object types were generated for a total of 5968 objects, which details are depicted in Figure 11. In this case, we can notice the object *transaction hash* for each event, as it is present in all transactions, while there is a unique *contract address* since all extractions are related to the same contract, and 624 different *senders*. Other highly occurring objects are *spender* and *amount*, derived from the function inputs, while *approval* and *transfer* are blockchain events. Finally, the analysed contract shows a high number of interactions with other smart contracts as demonstrated by the occurrence of internal transaction objects.

A more detailed distribution of objects over events is reported in Figure 12. The major occurrence of objects is related to the *approve* event as it is the most frequent activity. Peculiar is instead the behaviour of internal transactions, which occur only in the *sendFrom* event. This happens since the *sendFrom* function serves to transfer tokens with complex logic, such as calculating fees and involving many cross-chain messages.

The precise details about the created relationships are shown in Figure 13.

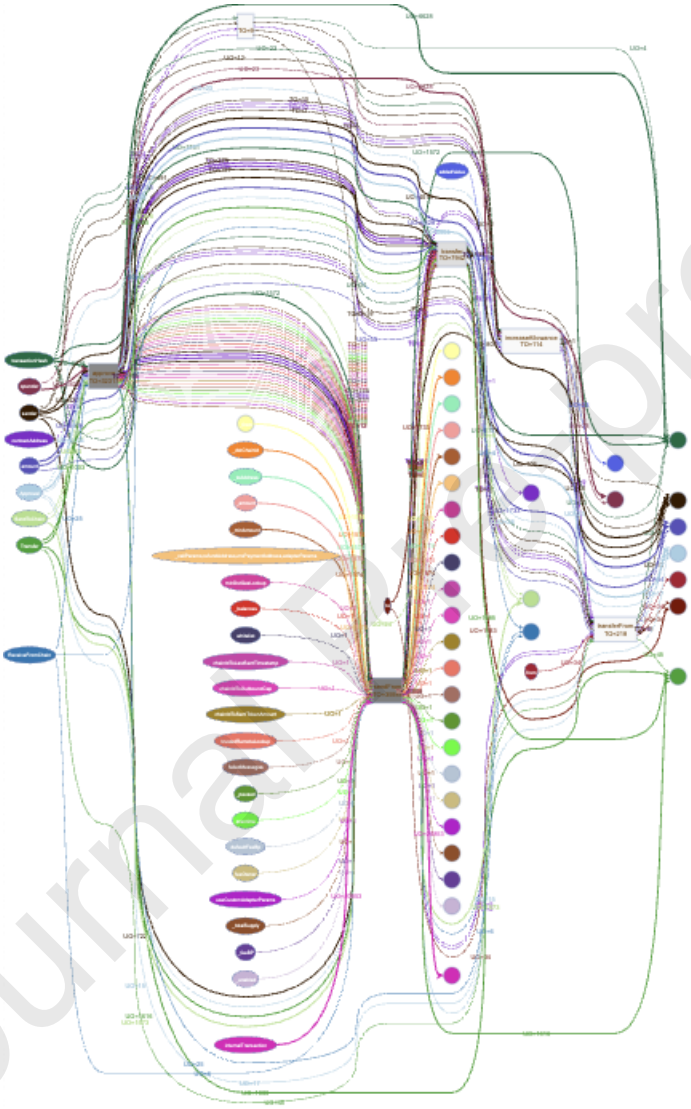


Figure 9: Pancakge Graph OCEL

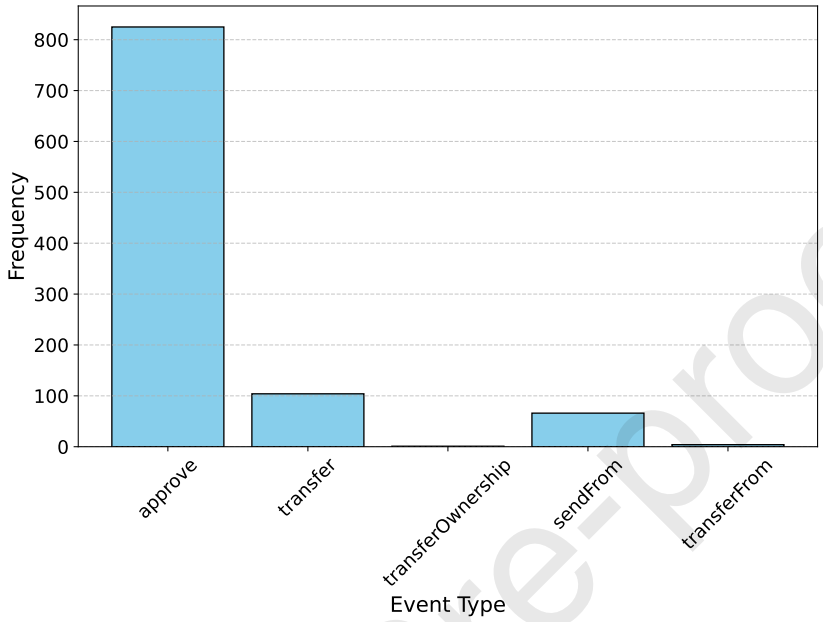


Figure 10: Number of occurrences for events

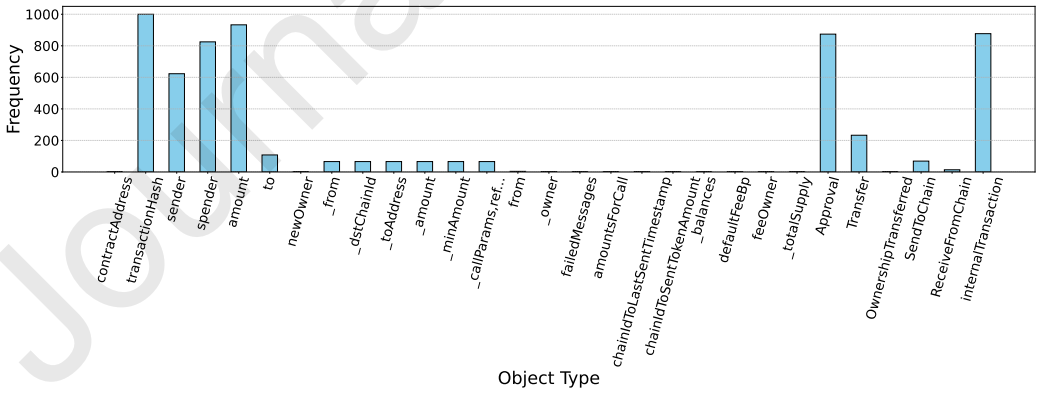


Figure 11: Number of occurrences for objects

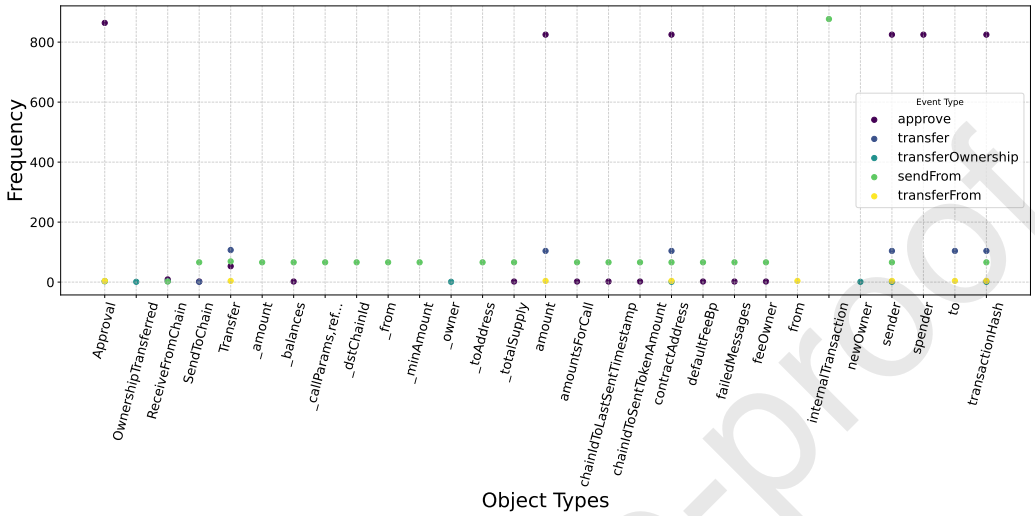
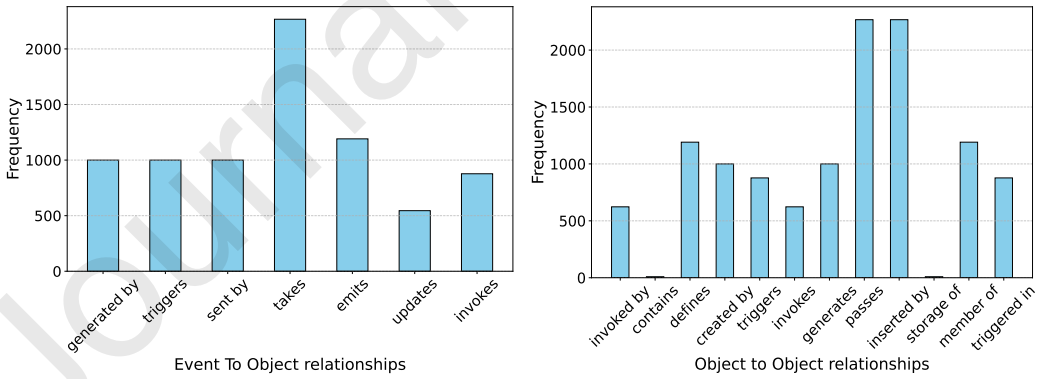


Figure 12: Distribution of objects over events



(a) Occurrence of event-to-object relationships (b) Occurrence of object-to-object relationships

Figure 13: Occurrence of relationships in the log

In the case of E2O reported in Figure 13a, we can notice that the *takes* relationship is the most frequent one with 2267 occurrences. This is because the majority of objects in the log derive from blockchain inputs and are consequently related to the event with the *takes* qualifier. This behaviour is also found in the O2O relationships depicted in Figure 13b. Also in this case, the *passes* and *inserted by* relationships occur 2267 times, connecting the *sender* object with the various input objects and vice versa.

Cryotikitties. The second considered scenario is Cryptokitties[7], a blockchain-based game built on the Ethereum network that allows users to collect, breed, and trade unique digital cats. Each CryptoKitty has a distinct set of traits determined by a genetic algorithm, and players can breed their kitties to create new ones with unique appearances. The game includes a built-in marketplace where users can buy and sell kitties, making it both a collectible and interactive experience. This application has been widely considered in different approaches such as [15, 25, 26]. From this smart contract, we extracted 10991 transactions within the block range 4605182 - 20671393, having 16 different event types, 10991 events, 21 different object types, and 172788 objects. Figure 14 depicts the resulting DFG. Setup calls (*setCEO*, *setCFO*, *setSaleAuctionAddress*, *setSiringAuctionAddress*, *setGeneScienceAddress*, *unpause*) each appear exactly once, matching the requirement that you must configure the CEO/CFO roles and register the auction contracts before you can list any kitties for sale or breeding. Notably, the discrepancy between the number of *breedWithAuto* (4,843) and *giveBirth* (10) calls could suggest a significant drop-off in user engagement during the breeding process and further investigation could be needed.

RotoHive. Finally, the third application is RotoHive[8], a fantasy sports platform where users collaboratively rank players each week. Participants submit their rankings, and top contributors are rewarded based on how well their predictions align with real-world outcomes. We considered this case since it was deeply explored in [27]. In particular, we extracted 3725 transactions from the block range 6247815 - 8678932, and after mapping all possibilities, we obtained 8 different event types, 3725 events, 19 different object types, and 21421 objects. The resulting DFG is visible in Figure 15. The obtained

[7] https://www.cryptokitties.co/
[8] https://www.rotohive.com

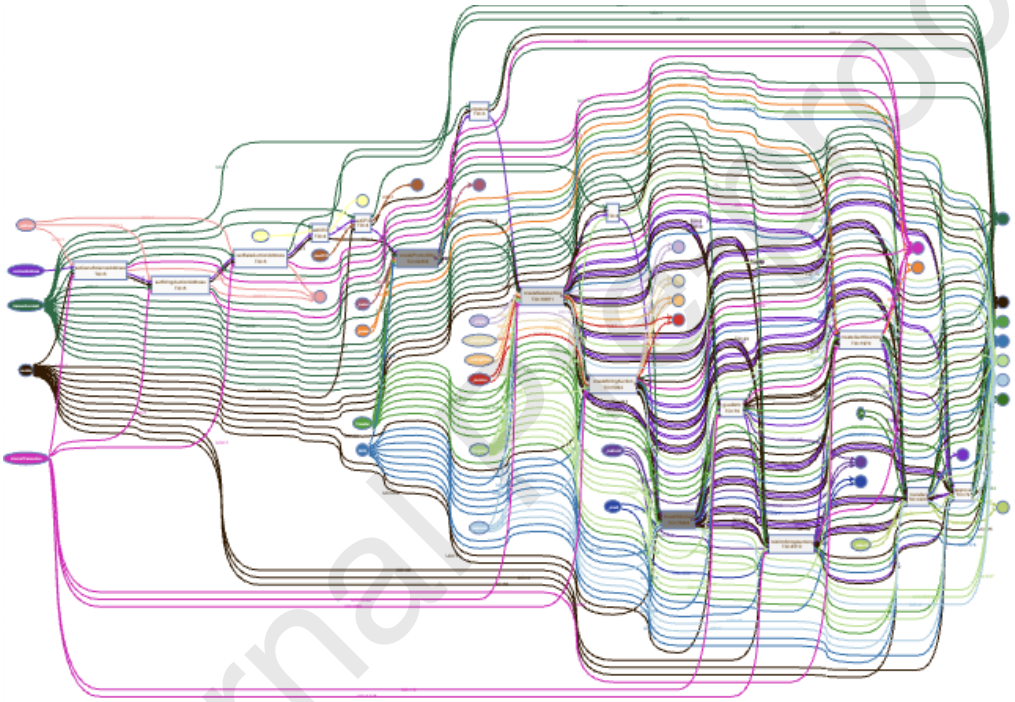


Figure 14: CryptoKitties Graph OCEL

graph reflects the lifecycle of the application given the presence of *setTokenContract* and *createTournament* and events at the beginning, alongside terminating events like *closeTournament* and *destroyRoto*. This outcome corresponds with the expected phases of initializing the tournament environment, executing staking and reward cycles, and closing tournaments. An interesting aspect is the high occurrences of *rewardRoto* events, highlighting the criticality of this phase. This matches the expected behaviour where reward calculation and distribution are key functions. The complexity and volume suggest events that might benefit from optimisation or specialised monitoring.

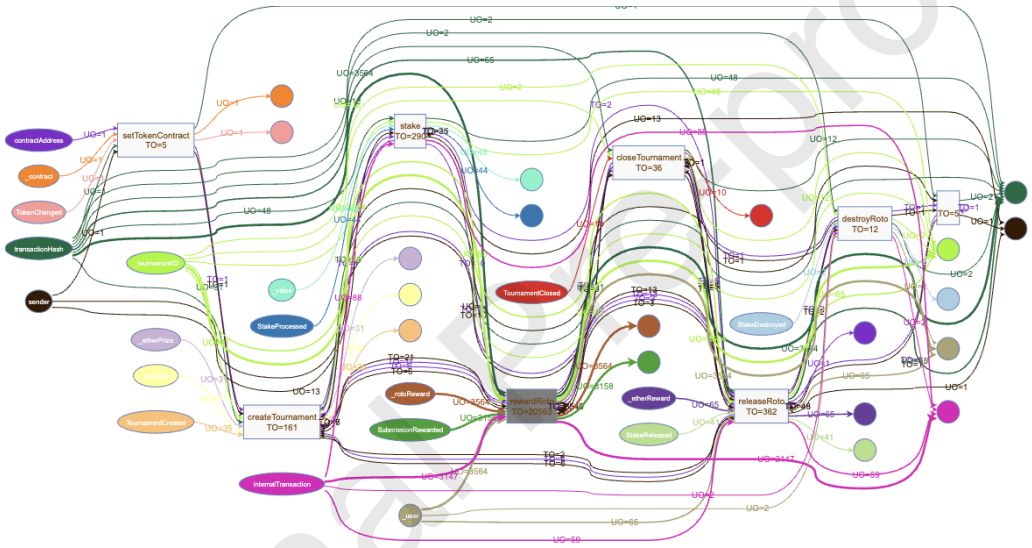


Figure 15: RotoHive Graph OCEL

5.2. *Performance analysis*

To further evaluate the mapping methodology, we analysed its performance in generating an OCEL 2.0 log. Specifically, we measured the time required to generate a log using different selection strategies in a scaling manner. We began by selecting only one object type and incrementally added one more at each step until all seven possible object types were included. For example, the first test involved selecting only the *contract address* as the object type, while the second included both *contract address* and *transaction hash*. In the final test, all object types were selected simultaneously. Additionally,

Table 3: Amount of time required for generating OCEL 2.0 log. CA = Contract Address, TH = Transaction Hash, SE = Sender, IN = Inputs, SV = State variables, EV = Events, IT = Internal Transactions. The "+" indicates that the object is included in the extraction with the previous objects (e.g., +TH = CA+TH, +SE = CA+TH+SE).

		Objects						
		CA	+TH	+SE	+IN	+SV	+EV	+IT
TXs	1000	0.01s	0.01s	0.03s	0.59s	0.64s	0.90s	1.15s
	2000	0.02s	0.03s	0.12	2.53s	2.71s	3.93s	5.51s
	3000	0.02s	0.03s	0.14	5.63s	6.03s	8.74s	12.54s
	4000	0.03s	0.06s	0.26	10.45s	11.24s	15.98s	23.27s
	5000	0.05s	0.09s	0.38	16.58s	17.48s	25.86s	39.51s
	6000	0.20s	0.17s	0.57	24.63s	25.82s	37.33s	55.48s
	7000	0.13s	0.17s	0.75	32.09s	34.30s	49.20s	76.14s
	8000	0.32s	0.23s	1.01	42.62s	46.40s	66.23s	97.93s
	9000	0.18s	0.30s	1.24	54.06s	58.66s	84.36s	124.63s
	10000	0.22s	0.38s	1.55	66.82s	72.22s	104.31s	156.05s

each selection strategy was evaluated over logs with an increasing number of transactions. For this purpose, we used the PancakeSwap smart contract, extracting 10,000 transactions from block 17283405 to block 18698323. In each experiment, we started with the first 1,000 transactions and gradually increased the dataset, up to the full log.

Table 3 presents the results of the experiments, reporting the time (in seconds) required to generate the log for each selection strategy. Figure 16 depicts these results, highlighting the trends as the log size increases. Analysing the outcomes, we observe that the first three mappings, including up to three object types (*CA*, *TH*, and *SE*), demonstrate the best performance, consistently remaining under 2 seconds. This behaviour is due to the relatively simple structure and low complexity of these data types, which do not require expensive computations for processing and mapping. A second group of results emerges when mapping four or five object types. In the case of four objects, the inclusion of *IN* leads to a noticeable increase in generation time, especially when processing 10,000 transactions. This is primarily because input objects have a more complex and nested structure, and since they appear in all transactions, their processing requires significantly more effort compared to previous cases. However, the addition of a fifth object type, *SV*, does not have a substantial impact on performance.

This is attributed to the design of the mapping for state variables, which generate unique objects updated over time. As a result, state variable object generation is less frequent and requires less computation. Finally, mapping six and seven object types through the first inclusion of *EV* and then also of *IT*, leads to further significant increases in generation time. This is due to the highly complex structure and frequent occurrence of events and internal transaction objects within the extracted dataset.

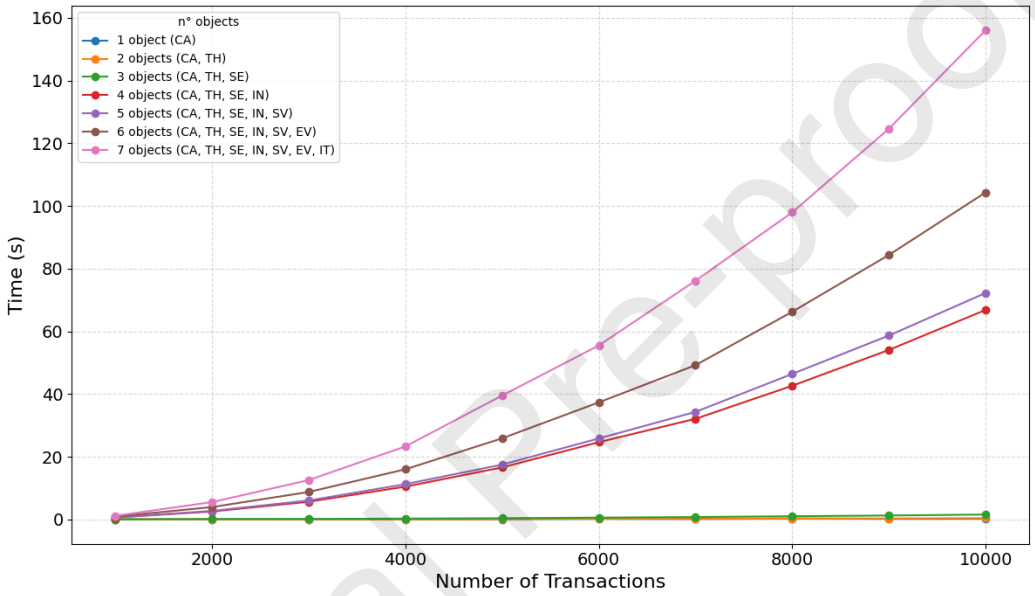


Figure 16: Trend of mapping performance

Overall, this evaluation provides an overview of the performance of the mapping methodology when applied to a real-world scenario. As expected, the complexity of the data structure in the blockchain log impacts performance due to the computational effort required for processing. However, it is important to note that performance may vary depending on the specific log being analysed. For instance, if a smart contract does not trigger a large number of internal transactions, its mapping would be faster than what was observed in this evaluation. This underlines that the performance of the proposed mapping approach is also dependent on the characteristics of the input log and the volume of data contained in each transaction.

5.3. Assessment with the Involvement of Practitioners

To evaluate the usability of the proposed methodology and the implemented application, we conducted an assessment involving practitioners. The objective of the experiment was to determine whether the provided mapping functionalities are usable by practitioners and whether they support the analysis of blockchain applications.

Experiment Set-Up. In setting up the experiment, we involved 15 Ph.D. candidates from the University of Camerino, all holding an MSc in Computer Science. They have a preliminary knowledge in blockchain technologies acquired through academic coursework and no technical expertise with process mining techniques. Currently, they are engaged in various research areas and the majority of them are interested in analysis activities.

In the experiments, the students were required to use the mapping functionalities and select various options to construct the final object-centric event log. Before the task, they were given an introduction to object-centric analysis and the OCEL 2.0 format. Additionally, all participants were provided with a shared blockchain event log, which structure and content were thoroughly explained in advance. At the end of the activity, the students were asked to complete the questionnaires provided in the Appendix, aimed at evaluating the implemented mapping methodology in terms of usability and perceived usefulness. The questionnaire was designed following the guidelines outlined in [28]. For each question, participants could select only one response that best reflected their experience using the application. To evaluate the results, we subsequently applied the System Usability Scale (SUS), which, as described in [29], "provides a quick and dirty, yet reliable, tool for measuring usability".

Experiment Results. After the students completed the experiment, we analysed the results of the questionnaires they had filled out. Table 4 presents the responses collected from all participants. The questionnaire design intentionally alternates between positively and negatively worded items. Specifically, for questions 1, 3, 5, 7, and 9, higher scores indicate more positive evaluations, while for questions 2, 4, 6, 8, and 10, lower scores are more favourable. This strategy is commonly used to reduce the likelihood of careless or patterned responses. Once all questionnaires were collected, the following formula was

applied to compute the overall usability score:

$$\mathcal{S} = \frac{\sum_{i=1}^{N}(\sum_{j=1}^{5}(5 - res_{i,2j}) + \sum_{j=0}^{4}(res_{i,2j+1} - 1))}{N} \times 2.5$$

It provides a way to get an overall score ($\mathcal{S}$) that can be used to globally assess the experiment results. In the formula, N represents the number of returned questionnaires, while $res_{i,j}$ is the response provided to the questionnaire by each participant (i) to every single question (j). The formula at first conducts all the answers to the scale 0 – 4, where positive answers now always correspond to higher values. Then, the total sum is computed, getting a number between 0 and 40, and the average over all the questionnaires is derived. Finally, the number is multiplied by 2.5 to get a final score $\mathcal{S}$ in the range 0 – 100. For the experiment we ran, the calculation of the SUS score gives us the value of 84.83. According to the proponents of the SUS approach, this is somehow a good result. Indeed, in their experience, values for $\mathcal{S}$ greater than 68 relate to perceived usability somewhat better with respect to other used software. Being the involved users somehow experienced with analysis techniques and tools, this suggests that overall they got a relatively positive experience in using the proposed mapping.

Going more in detail, we can observe that the best evaluation was obtained by question 5 ("*I would imagine that most people would learn to use the application very quickly.*") that got an average score of 4.9, while the worst evaluation was reported by question 4 ("*I think that I would need assistance to be able to use the application.*") with an average score of 2.13 (higher score indicates a worse evaluation). This highlights that the application appears to have an easy learning curve and is user-friendly, although an initial technical introduction is still considered beneficial. All the other evaluation exhibits a positive average, confirming the overall usability of the implemented methodology and its functionalities.

5.4. Discussion

Analysing the proposed methodology, some considerations can be made. The proposed mapping operations occur after execution, meaning they do not affect how smart contracts are deployed or how interactions are validated and stored on-chain. Thus, the method has no direct impact on the decentralisation of the underlying blockchain network.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Student 1	5	2	5	2	5	1	5	1	4	2
Student 2	4	2	4	2	4	2	5	1	4	2
Student 3	5	1	5	1	5	1	5	3	5	2
Student 4	4	2	5	4	3	3	5	2	4	1
Student 5	5	1	5	2	4	1	5	1	4	1
Student 6	4	1	5	1	4	1	5	1	5	1
Student 7	4	1	5	1	5	1	5	1	5	1
Student 8	4	1	4	3	4	1	5	1	4	2
Student 9	4	1	5	2	5	1	5	2	4	1
Student 10	5	1	5	1	5	1	5	2	5	5
Student 11	2	2	3	5	4	1	5	5	2	3
Student 12	4	1	4	2	4	1	4	1	4	2
Student 13	5	2	4	1	4	1	5	1	4	1
Student 14	4	2	4	4	4	2	5	2	5	2
Student 15	4	1	5	1	4	1	5	1	5	1
Avg	4.20	1.40	4.53	2.13	4.26	1.26	4.93	1.66	4.26	1.80

Table 4: Results from the questionnaires on usability related to the mapping methodology

A second important aspect is the generalisability of the methodology. The current implementation relies on EVM-based concepts and data that, however, are common in many different blockchains of other natures, despite they can have differences and peculiarities. Furthermore, the general design behind the mapping is that each smart contract function is mapped as an event, and that all data extracted from the corresponding transaction generates objects. This ensures that the methodology does not rely on specific assumptions, such as the presence of blockchain events in the log. While defining blockchain events is common, it depends on the smart contract developer's choices and may not always be present. In this setting, the inclusion of a different blockchain paradigm does not require the revision of the conceptual mapping unless it brings a completely different structure. Also, from a technical perspective, the mapping algorithm works on the expected entries of a JSON log, thus it is easily extensible to support a different structure. A future work could be to integrate a generic algorithm that accepts a generic JSON log, permitting the abstraction from EVM-blockchains' technicalities and data formats.

6. Related Works

The analysis of blockchain applications has always been a topic of interest, from the advent of Bitcoin to the growth of Ethereum. To this end, solutions have been proposed for blockchain data analytics and visualisation [30, 31, 5, 32]. These works can be classified into different areas of interest, mostly related to illegal activity detection, financial analysis, user analysis, and more [30]. This is done by highlighting and revealing specific perspectives of an application, like value tracing of tokens, activities of suspicious accounts, and the time analysis of transactions. The main objective is to support experts during analysis activities, providing relevant insights for detecting anomalies and informed decision-making.

Recently, process mining techniques have been recognised as a proper solution to analyse blockchain-based applications to enhance transparency and check runtime data [3]. Indeed, in a blockchain application, smart contracts can be seen as processes, encoding the business logic which is executed in a certain order and produces on-chain records. When using process mining techniques, this information can be used to enable the auditing of the application, revealing the actual execution flow, detecting variations, unexpected behaviour, and adherence to predefined rules [27]. In this direction, many works demonstrated the potential of process mining for blockchain applications, proposing frameworks and solutions to enable the analysis of blockchain data and dealing with the complexity and peculiarities of the domain [27, 33, 26, 34, 35, 36, 25]. In particular, several effort was made in the extraction and preprocessing tasks [8], critical during process mining projects, typically requiring more than 80% of resources [17]. When considering blockchain technology, extensive extraction is indeed required due to the structure of the ledger and the encoded nature of data. Furthermore, retrieved data does not match process mining formats, thus requiring additional preprocessing to obtain usable event logs. In the following, we describe the first proposals, highlighting the main objective and features.

In [27], the ABC methodology is introduced for auditing smart contracts of Ethereum-based blockchain applications using process mining. The methodology begins with the extraction of smart contract transactions for analysis. An XES event log is then automatically generated by clustering transactions that share the same sender and contain the executed functions. The methodology was applied to the RotoHive case study, where discovery techniques were used to derive process models representing the smart

contract execution. In the final phase, the generated models are analysed to assess their conformance with expected behaviours. In [33], the authors propose a framework to enable the auditing of blockchain-based business processes. To this end, the framework extracts Ethereum execution data and maps it to the XES standard, producing a corresponding event log. To demonstrate the framework in action, an Incident Management process is used. Its log is analysed within a process mining framework to discover the process model and check its conformance with the original one. However, this work specifically focuses on a blockchain-based BPM execution engine, making certain assumptions, and does not support generic blockchain applications. [26] presents a framework for extracting execution data from Ethereum blockchain applications, converting it into the XES format, and generating logging functionalities for smart contracts. Unlike previous works, the framework introduces a customizable strategy based on a manifest, which can be configured to specify the smart contracts and transactions to extract, as well as how to map blockchain concepts to XES and add custom extensions. The framework was evaluated using the CryptoKitties case study, where discovery techniques were applied to visualise the various behaviours emerging from the generated logs. The Blockchain Logging Framework (BLF) is proposed in [34]. BLF extends the work presented in [35], enabling the extraction of data from multiple blockchains (predefined support for Ethereum and Hyperledger Fabric) and formatting it in various output formats, including CSV and XES. To allow flexible customisation, a manifest must be configured, specifying the target blockchain, extraction filters, and data processing rules. BLF was evaluated using the HyperKitties application, a custom implementation of CryptoKitties on Hyperledger Fabric. [35] focuses on the Augur decentralised application to demonstrate the potential of process mining for blockchain. To this end, the approaches proposed in [26, 34] were used to extract blockchain events from the Ethereum-based Augur smart contract and convert them into an XES event log. This log was then used to apply discovery and conformance checking techniques. The results of the analysis provided valuable insights into the behaviours of Augur, confirming the usefulness of process mining in this context. [36] proposes an application for extracting complex events from blockchain systems. The work specifically focuses on Hyperledger Fabric, Hyperledger Composer, and the IBM Blockchain Platform, as these technologies are commonly used in business applications. The extracted data is ultimately converted into a CSV file, making it suitable for use in subsequent process mining analyses.

The application was evaluated using a vehicle manufacturing smart contract. The Evelog methodology is presented in [25]. Evelog enables the extraction of data from EVM-based blockchains, offering dedicated functionalities for handling internal transactions. Transactions can be extracted from multiple smart contracts, which can be specified with minimal configuration. A selection step allows users to define how traces should be constructed for the event log, which is ultimately generated in the XES format. Evelog was demonstrated on the CryptoKitties application, where various discovery techniques were applied to the extracted data.

The aforementioned works represent the first attempts and solutions to apply process mining on blockchain. Nevertheless, despite the various challenges faced, all of them are tailored for traditional process mining analysis employing the XES standard, thus not making them suitable for object-centric analysis. Indeed, object-centric process mining has recently gained significant interest in discovering systems from novel perspectives. In this context, some approaches have been proposed to apply object-centric analysis to blockchain applications, enabling a more comprehensive understanding of their processes and interactions within decentralised ecosystems.

In [15] the Artifact-Centric Event Log (ACEL) logging format is proposed. ACEL extends OCEL to capture artefacts (business entities represented in objects), express their relationships and track their lifecycle (i.e., changes in the state). To achieve this, the authors propose a methodology to configure, extract, and generate ACEL logs based on blockchain data, focusing on extracting blockchain events. This approach also involves a configuration phase requiring the manual definition of artefacts, relationships and the mapping rules to convert extracted blockchain events to ACEL elements. While this methodology provides novel artefacts-oriented elements and concepts, with the introduction of OCEL 2.0, new features such as support for object lifecycles are available, making it usable for a broader range of usages. [16] presents an approach for using object-centric analysis of Decentralised Applications (DApps) to retrieve and decode blockchain data with OCEL. This work focuses on the Augur DApp, extracting and analysing contract creations, creation sub-trees, and relevant execution data, including message calls and events. The results demonstrate the potential of object-centric approaches for analysing DApps. Also in this case, the OCEL 2.0 specification and its extensions could provide novel perspectives by enhancing expressiveness, which needs further exploration. Additionally, this approach does not place a strong emphasis on the mapping phase and

Table 5: Table of identified related works and their characteristics

Work	Mapping perspective	Custom mapping support	Technical effort required	Output format
[27]	Function-based	No	-	XES
[33]	Function-based	No	-	XES
[26]	Event-based	Yes	Manifest configuration	XES
[35]	Event-based	Yes	Manifest configuration	XES
[34]	Event-based	No	-	XES
[36]	Event-based	No	-	CSV
[25]	Selectable	Yes	Selection of: Case id, activity, timestamp	XES
[16]	Event-based	No	-	OCEL
[15]	Event-based	Yes	Configuration of: artefacts, mapping rules, relationships	ACEL
[37]	Function-based, Event-based, Contract creation	Yes	Mapping file	OCEL 2.0
This work	Function-based	Yes	Object selection	OCEL 2.0

does not provide explicit functionalities for custom strategies. The work was extended in [37] to support a richer extraction of data and its mapping to the novel OCEL 2.0 standard. Specifically, the approach aims to capture the behaviour of dynamically deployed DApps by leveraging OCEL 2.0 to represent the involved components, their relationships, and their lifecycles. To this end, an extraction phase retrieves *contract creations*, *function calls* (including delegatecalls), and emitted *blockchain events*, capturing the hierarchy of smart contracts and their interactions within the DApp. These three types of data are then mapped to OCEL 2.0 events to generate the final event log. Each event has attributes such as from, to, gas, gasUsed, value, input, output, error, transactionHash, transactionIndex, blockNumber, and additional parameters specific to function calls, delegatecalls, and blockchain events. For event-to-object (E2O) relationships, domain-specific qualifiers are derived from the parameters of calls and blockchain events. Object defi-

nitions, on the other hand, require manual configuration through a mapping file. While some objects and attributes can be automatically inferred, others are dependent on the specific domain and smart contract logic, necessitating manual specification. The approach was evaluated using the Augur application. Although this work represents the first attempt to model blockchain applications using the OCEL 2.0 standard, it differs from our contribution in several key aspects. The presented work focuses specifically on DApps and their dynamic creation and interaction of smart contracts, offering a dedicated perspective for this specific analysis. In contrast, our methodology does not target this specific perspective, but it is designed to be more general, to enable the representation of a wide range of meaningful information about blockchain applications. Regarding the types of data considered, [37] offers a more fine-grained view of attributes and parameters related to internal transactions. In contrast, our approach focuses on the inclusion of state variables, providing a complementary perspective on the execution data. Finally, the representation of data in an object-centric format follows a different logic. The mapping proposed in [37] is driven by the goal of explicitly capturing specific operations (contract creations, function calls/delegatecalls, and emitted events), centring the mapping around them. Furthermore, the definition of objects in that approach strictly depends on the domain of the DApp and requires manual configuration by the user. In contrast, our approach aims to provide a more general and automated mapping strategy that does not depend on domain-specific and manual customisations. By doing so, we aim to simplify the generation of object-centric logs and provide a usable solution. This design choice is also reflected in our object mapping strategy, which allows objects to be selected from any data type, facilitating a more flexible and comprehensive representation. l

Table 5 provides a resuming overview of current approaches, highlighting the main aspects related to mapping from blockchain data to process mining logs. Different from the aforementioned works, our objective is to enable object-centric analysis of blockchain applications, reducing the specific and technical knowledge required and making it accessible to a wider audience. This is done through a mapping methodology that facilitates the creation of OCEL 2.0 logs from any smart contract, simplifying the transition from blockchain data to object-centric concepts. Our methodology ensures compatibility with existing tools and techniques, enabling comprehensive analysis of blockchain applications employing the object-centric paradigm. Another significant difference is the interpretation of object-centric events over

blockchain data. Presented approaches primarily consider blockchain events, mapping them to object-centric events. Differently, our methodology focuses on smart contract functions, offering a distinct perspective on the execution of applications and their associated smart contracts.

7. Conclusions

Blockchain enables applications where interactions and business logic execute autonomously, relying on immutable and decentralised infrastructure. Smart contracts play a central role by encoding behaviours, enforcing constraints, and generating immutable execution data. This data offers valuable insights into application behaviour, revealing anomalies, dependencies, and optimisation opportunities. Process mining techniques have been recently applied to analyse blockchain execution, but they need to deal with the complexity and heterogeneity of blockchain data. Object-centric process mining addresses these limitations by capturing relationships between multiple interacting entities, offering a more expressive and comprehensive analysis. However, applying object-centric process mining to blockchain presents new challenges, such as defining appropriate mapping strategies, handling different levels of granularity, and selecting standardised formats to ensure compatibility with existing tools.

To tackle these challenges, we propose a mapping methodology that enables the applicability of object-centric analysis. This is done by providing mapping capabilities to pass from blockchain data to object-centric logs, usable with current existing tools and techniques. Implemented as a web application, our methodology supports the mapping process, making it accessible to a broader audience without requiring technical expertise or complex manual configurations. To show the effectiveness of our methodology, we applied it to the PancakeSwap blockchain application, generating an object-centric event log and performing an analysis using established process mining tools.

In future directions, we intend to support a fully customisable selection strategy which allows users to define the mapping of blockchain data to object-centric elements. In this way, the selection does not need to be bound to a specific starting format and can be expanded to any blockchain log. We also plan to provide an accessible ecosystem where extracted data is published and made accessible through a database. In this way, we aim that interested researchers can collaborate and share meaningful logs to test different approaches and functionalities. Finally, we plan to integrate func-

tionalities for the log analysis to provide a first overview of created elements and relationships.

Acknowledgment

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

Appendix A - Questionnaire

We report here the System Usability Scale (SUS) questionnaires issued to students which results are discussed in Section 5.

Question 1 - I think that I would like to use the application frequently.

1	**2**	**3**	**4**	**5**
Strongly disagree				Strongly agree

Question 2 - I found the application unnecessarily complex.

1	**2**	**3**	**4**	**5**
Strongly disagree				Strongly agree

Question 3 - I thought the application was easy to use.

1	**2**	**3**	**4**	**5**
Strongly disagree				Strongly agree

Question 4 - I think that I would need assistance to be able to use the application.

1	**2**	**3**	**4**	**5**
Strongly disagree				Strongly agree

Question 5 - I found the various functions in the application were well integrated.

1	**2**	**3**	**4**	**5**
Strongly disagree				Strongly agree

Question 6 - I thought there was too much inconsistency in the application.

1	**2**	**3**	**4**	**5**
Strongly disagree				Strongly agree

Question 7 - I would imagine that most people would learn to use the application very quickly.

1	**2**	**3**	**4**	**5**
Strongly disagree				Strongly agree

Question 8 - I found the application very cumbersome/awkward to use.

1	**2**	**3**	**4**	**5**
Strongly disagree				Strongly agree

Question 9 - I felt very confident using the application.

1	**2**	**3**	**4**	**5**
Strongly disagree				Strongly agree

Question 10 - I needed to learn many things before I could get going with the application.

1	**2**	**3**	**4**	**5**
Strongly disagree				Strongly agree

References

[1] C. Di Ciccio, G. Meroni, P. Plebani, Business process monitoring on blockchains: Potentials and challenges, in: Enterprise, Business-Process and Information Systems Modeling, Vol. 387 of LNBIP, Springer, 2020, pp. 36–51.

[2] C. Di Ciccio, G. Meroni, P. Plebani, On the adoption of blockchain for business process monitoring, Software and Systems Modeling 21 (3) (2022) 915–937. doi:10.1007/S10270-021-00959-X.

[3] R. Hobeck, C. Klinkmüller, H. D. Bandara, I. Weber, W. van der Aalst, On the suitability of process mining for enhancing transparency of blockchain applications, Business & Information Systems Engineering (2024) 1–20.

[4] G. Ibba, S. Aufiero, S. Bartolucci, R. Neykova, M. Ortu, R. Tonelli, G. Destefanis, Mindthedapp: a toolchain for complex network-driven structural analysis of ethereum-based decentralised applications, IEEE Access (2024).

[5] N. Tovanich, N. Heulot, J.-D. Fekete, P. Isenberg, Visualization of blockchain data: a systematic review, IEEE transactions on visualization and computer graphics 27 (7) (2019) 3135–3152.

[6] W. M. P. van der Aalst, Process mining: A 360 degree overview, in: W. M. P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, Vol. 448 of Lecture Notes in Business Information Processing, Springer, 2022, pp. 3–34.

[7] W. M. P. van der Aalst, Process Mining - Data Science in Action, Second Edition, Springer, 2016.

[8] L. Moctar-M'Baba, M. Sellami, W. Gaaloul, M. F. Nanne, Blockchain logging for process mining: a systematic review, in: International Conference on System Sciences, ScholarSpace, 2022, pp. 1–10.
URL http://hdl.handle.net/10125/80091

[9] C. W. Gunther, H. Verbeek, Xes-standard definition (2014).

[10] W. M. van der Aalst, Object-centric process mining: Dealing with divergence and convergence in event data, in: International Conference on Software Engineering and Formal Methods, Springer, 2019, pp. 3–25.

[11] J. Mendling, I. Weber, W. V. D. Aalst, J. V. Brocke, C. Cabanillas, F. Daniel, S. Debois, C. D. Ciccio, M. Dumas, S. Dustdar, et al., Blockchains for business process management-challenges and opportunities, ACM Transactions on Management Information Systems (TMIS) 9 (1) (2018) 1–16.

[12] H.-Y. Paik, X. Xu, H. D. Bandara, S. U. Lee, S. K. Lo, Analysis of data management in blockchain-based systems: From architecture to governance, Ieee Access 7 (2019) 186091–186107.

[13] D. Fahland, M. Montali, J. Lebherz, W. M. van der Aalst, M. van Asseldonk, P. Blank, L. Bosmans, M. Brenscheidt, C. di Ciccio, A. Delgado, et al., Towards a simple and extensible standard for object-centric event data (oced)–core model, design space, and lessons learned, arXiv preprint arXiv:2410.14495 (2024).

[14] A. F. Ghahfarokhi, G. Park, A. Berti, W. M. van der Aalst, Ocel: a standard for object-centric event logs, in: European Conference on Advances in Databases and Information Systems, Springer, 2021, pp. 169–175.

[15] L. Moctar-M'Baba, N. Assy, M. Sellami, W. Gaaloul, M. F. Nanne, Process mining for artifact-centric blockchain applications, Simulation Modelling Practice and Theory — Journal 127 (2023) 102779. doi:https://doi.org/10.1016/j.simpat.2023.102779.

[16] R. Hobeck, I. Weber, Towards object-centric process mining for blockchain applications, in: Business Process Management: Blockchain, Robotic Process Automation and Educators Forum, Vol. 491 of LNBIP, Springer, 2023, pp. 51–65.

[17] K. Diba, K. Batoulis, M. Weidlich, M. Weske, Extraction, correlation, and abstraction of event data for process mining, WIREs Data Mining Knowl. Discov. 10 (3) (2020).

[18] P. Zheng, Z. Jiang, J. Wu, Z. Zheng, Blockchain-based decentralized application: A survey, IEEE Open Journal of the Computer Society 4 (2023) 121–133.

[19] R. Jia, S. Yin, To evm or not to evm: Blockchain compatibility and network effects, in: Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security, 2022, pp. 23–29.

[20] A. Berti, I. Koren, J. N. Adams, G. Park, B. Knopp, N. Graves, M. Rafiei, L. Liß, L. T. G. Unterberg, Y. Zhang, et al., Ocel (object-centric event log) 2.0 specification, arXiv preprint arXiv:2403.01975 (2024).

[21] V. Buterin, Ethereum: A next-generation smart contract and decentralized application platform., https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf (2014).

[22] F. Corradini, A. Marcelletti, A. Morichetta, B. Re, A methodology for extracting and decoding smart contracts data, Computer Communications (2025) 108204.

[23] F. Corradini, A. Marcelletti, A. Morichetta, B. Re, A data extraction methodology for ethereum smart contracts, in: International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events, IEEE, 2024, pp. 524–529. doi:10.1109/PERCOMWORKSHOPS59983.2024.10502604.

[24] A. Berti, W. M. van der Aalst, Oc-pm: analyzing object-centric event logs and process models, International Journal on Software Tools for Technology Transfer 25 (1) (2023) 1–17.

[25] A. Morichetta, Y. Paoloni, B. Re, Event log extraction methodology for ethereum applications, Future Generation Computer Systems 164 (2025) 107566.

[26] C. Klinkmüller, A. Ponomarev, A. B. Tran, I. Weber, W. M. P. van der Aalst, Mining blockchain processes: Extracting process mining data from blockchain applications, in: Business Process Management: Blockchain and Central and Eastern Europe Forum, Vol. 361 of LNBIP, Springer, 2019, pp. 71–86. doi:10.1007/978-3-030-30429-4_6.

[27] F. Corradini, F. Marcantoni, A. Morichetta, A. Polini, B. Re, M. Sampaolo, Enabling auditing of smart contracts through process mining,

From Software Engineering to Formal Methods and Tools, and Back 11865 (2019) 467–480. doi:10.1007/978-3-030-30985-5_27.

[28] J. Brooke, et al., Sus-a quick and dirty usability scale, Usability evaluation in industry 189 (194) (1996) 4–7.

[29] A. Bangor, P. T. Kortum, J. T. Miller, An empirical evaluation of the system usability scale, Intl. Journal of Human–Computer Interaction 24 (6) (2008) 574–594.

[30] M. Bühlmann, H.-G. Fill, S. Curty, Blockchain data analytics: A scoping literature review and directions for future research, arXiv preprint arXiv:2505.04403 (2025).

[31] F. Härer, H.-G. Fill, A comparison of approaches for visualizing blockchains and smart contracts, Jusletter IT (2019).

[32] J. Song, P. Zhang, Q. Qu, Y. Bai, Y. Gu, G. Yu, Why blockchain needs graph: A survey on studies, scenarios, and solutions, Journal of Parallel and Distributed Computing 180 (2023) 104730.

[33] R. Mühlberger, S. Bachhofner, C. D. Ciccio, L. García-Bañuelos, O. López-Pintado, Extracting event logs for process mining from data stored on the blockchain, in: Business Process Management Workshops, Vol. 362 of LNBIP, Springer, 2019, pp. 690–703. doi:10.1007/978-3-030-37453-2_55.

[34] P. Beck, H. Bockrath, T. Knoche, M. Digtiar, T. Petrich, D. Romanchenko, R. Hobeck, L. Pufahl, C. Klinkmüller, I. Weber, BLF: A blockchain logging framework for mining blockchain data, in: Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Track at Business Process Management, Vol. 2973 of CEUR, CEUR-WS.org, 2021, pp. 111–115.

[35] R. Hobeck, C. Klinkmüller, H. M. N. D. Bandara, I. Weber, W. M. P. van der Aalst, Process mining on blockchain data: A case study of augur, in: Business Process Management: International Conference, Vol. 12875 of LNCS, Springer, 2021, pp. 306–323. doi:10.1007/978-3-030-85469-0_20.

[36] A. Koschmider, F. Duchmann, Extraction of meaningful events for process mining from blockchain, in: Blockchain and Robotic Process Automation, Springer, 2021, pp. 13–29.

[37] R. Hobeck, A. Berti, I. Weber, W. van der Aalst, Object-centric process mining for blockchain applications: Extracting and representing ethereum execution data in ocel 2.0, Enterprise Modelling and Information Systems Architectures (EMISAJ) 20 (2025).

Declaration of Interest Statement

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The author is an Editorial Board Member/Editor-in-Chief/Associate Editor/Guest Editor for this journal and was not involved in the editorial review or the decision to publish this article.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Flavio Corradini: Supervision, Funding acquisition, Project administration, Conceptualization.
Alessandro Marcelletti: Conceptualization, Investigation, Writing- Original draft, Data Curation, Validation.
Andrea Morichetta: Conceptualization, Investigation, Writing- Reviewing and Editing, Data Curation.
Barbara Re: Supervision, Conceptualization, Investigation, Writing- Reviewing and Editing.
Lorenzo Verducci: Data curation, Writing- Original draft, Software, Validation.