EasyChair Preprint

№ 10631

Towards Object-Centric Process Mining for Blockchain Applications

Richard Hobeck and Ingo Weber

July 29, 2023

Towards Object-centric Process Mining for Blockchain Applications

Richard Hobeck[1] and Ingo Weber[2]

[1] Chair of Software and Business Engineering, Technische Universitaet Berlin, Germany, first.last@tu-berlin.de
[2] Technical University of Munich, School of CIT, and Fraunhofer Gesellschaft, Munich, Germany, first.last@tum.de

Abstract. Process mining event logs are traditionally formatted to reflect the execution of a collection of individual process instances, with a fixed case notion. In practice, process instances are often intertwined, and the scope of a particular process is less static. When flattening complex application data to traditional event log formats, like XES, problems such as divergence and convergence occur in the resulting event logs. Object-centric process mining with its object-centric event log (OCEL) format were introduced to tackle these issues, supporting multiple case notions with a single event log. While the adoption of object-centric logging is starting to gain momentum in several domains, the use of OCEL for event logs of blockchain applications has seen little research. In this paper, we investigate blockchain data structures and map them to OCEL logging capabilities. We present an approach to extracting data from blockchain applications that requires little domain knowledge. We discuss how to map data items to fit object-centric event logs and provide and analyze a corresponding OCEL event log for a blockchain application. The approach is evaluated based on a use case and contrasted to a previous case study.

Key words: process mining, OCEL, blockchain.

1 Introduction

Blockchain is a distributed ledger that allows for computer-based coordination between parties in an otherwise trustless environment [21, p. 3]. Second generation blockchains (e.g., Ethereum) can act as execution engines for arbitrary code. Applications that are implemented on blockchains (*decentralized applications*, short: DApps) generate data during runtime that can be made subject to data analysis. However, blockchain data may be fragmented and encrypted, and accounts and keys may change over time: properties that pose challenges to analyzing blockchain data, e.g., with process mining techniques [14].

Process mining is a set of techniques to generate knowledge from process data [10]. As process mining is heavily dependent on data, different data exchange formats have been discussed to achieve interoperability between data producing and data consuming systems. Widely-adopted as a format for event

logs and an input format for process mining tools is the *eXtensible Event Stream* (XES) thanks to an early proclamation as an event log format standard [8]. XES allows only a single case notion which can cause problems in the event log, e.g., divergence and convergence [3]. To overcome these problems, the more recently established *Object-Centric Event Log* (OCEL) standard allows multiple case notions in one event log [7].

To date, XES remains the most widely-used standard for process mining event logs for blockchain applications [4]. Several approaches to creating XES event logs from blockchain data for process mining have been proposed. An overview of the existing approaches was presented in a recent systematic literature review [13]. [11] focuses on creating single-case notion event logs from blockchain log entries that are specified in smart contract code and written onto the blockchain. Their tool has been developed further and made blockchain platform independent [5]. [6] used fine-grain execution data of transactions that were generated by Ethereum nodes. They grouped the data by their sender address to create a case notion. Similarly, [16] decoded smart contract function calls and used names of called functions as activity names in event logs. [18] introduced steps to analyze the sojourn time between submissions of transactions and their inclusion in a block. There have also been initiatives to make blockchain event logs less dependent on a single-case notion. [12] extracted log entries for the Ethereum DApp *Cryptokitties* and presented a "artifact-centric" logging format based on an OCEL extension. The extension, however, has no process mining tool support. Current tools that create XES logs from blockchain data attempt to deal with data fragmentation and account changes by limiting the logging to certain accounts and event types impeding to document the full extent of a DApp's behavior.

In this paper, we address the above limitations with a two-pronged approach, specifically object-centric process mining for DApps and more comprehensive data capture. As such, we explore using the OCEL logging format to incorporate various types of execution data of an application in a blockchain environment in a single event log. The approach is aimed at suiting dynamic deployments and distributed logging, as commonly seen in blockchain applications. We make the following contributions: (1) we propose an object-centric approach to retrieving and decoding blockchain application data, (2) we broaden the set of considered data sources to include, among others, function calls and input data, logged variable values, dynamically created smart contracts and application structure, and tracking digital assets including cryptocurrency and (non-)fungible tokens (FTs and NFTs); (3) we conduct an initial evaluation of the proposed approach to investigate technical feasibility and comparatively analyze the resulting OCEL-log with an earlier case study that was based on XES.

We argue that challenges in creating event logs for blockchain applications can effectively be tackled using object-centric logging formats. In this paper, we explore particularities of data structures in a blockchain environment and map them to the capabilities of OCEL. We describe a data extraction method that flexibly responds to changes in a DApp's structure and updates of application

code. For an initial validation of the approach, we conduct an assertion [22] as a basis for future experiments.

The remainder of the paper is structured as follows: In Sec. 2 we describe the XES and OCEL logging formats. Sec. 3 describes data attributes that relate to execution data in a blockchain environment (specifically Ethereum) and that may be part of a comprehensive event log of a DApp. Our approach to extracting blockchain data is presented in Sec. 4. This is succeeded by presenting extracted data for the DApp *Augur*, and analysis and an evaluation of the approach in Sec. 5. The evaluation compares data availability and process mining insights with a case study performed on the same DApp with a single-notion event log. We discuss the findings of the paper in Sec. 6 and conclude the paper in Sec. 7.

2 Logging Formats in Process Mining

For every process mining endeavor, which aims to derive insights from process data, the data is the foundation. To ensure interoperability between different information systems (e.g., systems producing event data as output and systems consuming event data as input for process mining) a standardized exchange format called *eXtensible Event Stream* (XES) was agreed on in 2010 and defined as a standard in 2016 [1, 2]. XES is an instantiation or dialect of the eXtensible Markup Language (XML). XES defines syntax and semantics for an event log. As its main ingredients, XES defines the notions of a log, traces, events, and their respective attributes [2]. A log can contain traces, which in turn comprise events. A trace represents a *single* process instance.

However, during actual process executions, events often belong to multiple cases. An often-cited example is that of an online store where a customer created multiple orders with multiple items each; the store might send all items in a single shipment, or each item separately; and the customer might receive one invoice, or one per order, or one per shipment. What should the case notion be based on: one case per shipment, order, invoice, or customer? For most answers, cases are intertwined. Deciding on a case notion and creating a corresponding XES log hence includes a step of *flattening the log*. Approaches to flatten the event log may result in some of the following three issues. (1) *Deficiency*: if an event does not have the chosen case notion it does not appear in the data. (2) *Convergence*: if an event relates to two distinct cases in the chosen case notion, it will appear duplicated in the data. (3) *Divergence*: events of two different cases and a shared third case may appear causally related although they are not [3, 1].

To address these problems a new standard was published in 2020: Object-centric Event Log (OCEL) [7]. Like XES, OCEL also comprises events and their attributes. A major innovation of OCEL allows events to refer to *multiple* case notions. This is achieved by introducing *objects* which represent "physical and informational entities composing business processes such as materials, documents, products, invoices, etc." [7, p. 4]. Objects and events have many-to-many relations in OCEL (see Fig. 1).

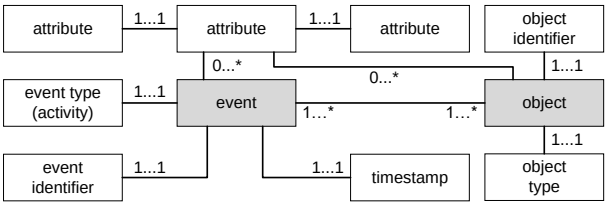


Fig. 1: UML diagram conceptualizing OCEL (adaption from [7]).

3 Data attributes for object-centric event logs from the Ethereum blockchain

Second-generation blockchains like Ethereum can be used as infrastructure for the execution of code that is deployed as smart contracts in *contract accounts* (CAs). Apart from CAs, a second type of accounts exists that are controlled by the private key holder (e.g., a user); these are called *externally owned accounts* (EOA)[1]. A blockchain is an append-only ledger. Data of the ledger is stored in consecutively created *blocks*. Each block is linked to its predecessor. Blocks contain *transactions* and *transaction receipts*, among others. Transactions capture transitions from one state of the blockchain to another [19]. The result of executing a transaction is hashed, and the hash is stored in the respective receipt; the receipt allows verifying that each machine that executes a transaction arrives at the same result, including CA log entries created as part of a contract invocation. Such log entries are used to communicate on-chain data as well as state changes to off-chain processing units.

On the Ethereum blockchain, transactions may contain invocations of smart contract functions, which result in computational steps being executed on the Ethereum Virtual Machine (EVM). These computational steps can be captured in transaction traces. Transaction traces exist on different levels of granularity. They can include assembly-level operations, e.g., comparisons and bit-wise logic operations, or push operations (within the EVM's computational stack); but they also comprise logging operations (emitting log entries) and system operations (e.g., creations of CAs and message calls between CAs) [19]. In contrast to earlier approaches, we here make use of relevant parts of traces. Fig. 2 is a class diagram depicting the relations between transactions, contracts, log entries, and traces.

Transaction traces exist temporarily and are not stored permanently in blocks of the Ethereum blockchain. In order to retrieve traces for historic transactions, transactions have to be replayed on the EVM (replay the transition from one state of the blockchain to the next) and the computational steps have to be stored separately from the blockchain. For that purpose, different tracers exist for the most widely used Ethereum execution client Geth[2,3].

[1] https://ethereum.org/en/developers/docs/accounts/, accessed 2023-05-15
[2] https://ethereum.org/en/developers/docs/nodes-and-clients/, accessed 2023-06-03
[3] https://geth.ethereum.org/docs/developers/evm-tracing/built-in-tracers, accessed 2023-05-16

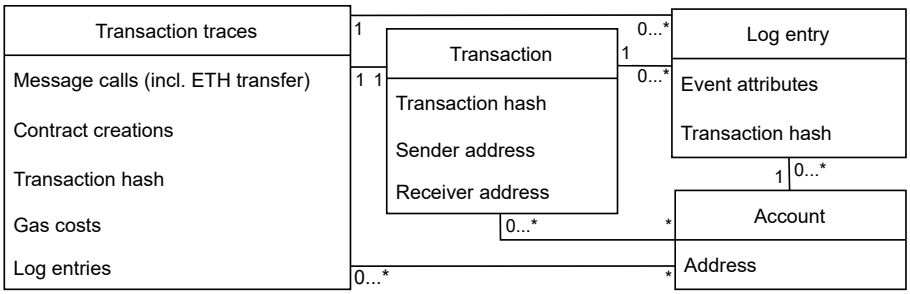


Fig. 2: Class diagram showing relations between contracts, transactions, events, and traces.

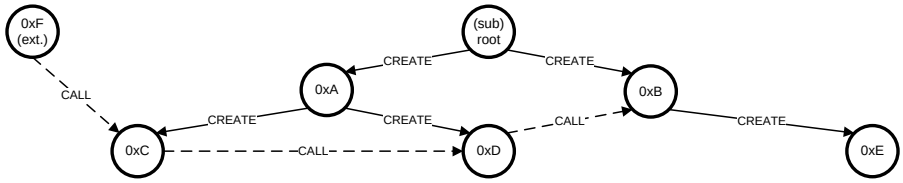


Fig. 3: Schema of a single tree-like deployment structure of smart contracts and possible message calls between accounts.

Depending on the objective of a **process analysis**, different data are relevant. In the context of user and contract behavior analysis as well as value stream analysis, from the sources above we considered the data listed in the following as relevant.

Contract creations: Knowing the accounts belonging to a DApp is essential to identify transactions that concern the DApp and contain user or application data. The creation of a CA is documented in a transaction trace with the mnemonic "CREATE" or "CREATE2" combined with other data attributes (e.g., creator and costs of the creation). The EVM interprets the accompanying input data and attempts to deploy it as smart contract code. CAs can be added to a DApp at initial deployment, but there are also mechanisms for updating or adding smart contracts to running DApps, e.g., the factory pattern, registry pattern [20], or diamond proxy[4]. A set of CAs belonging to one DApp that gets deployed can be seen as a tree structure (or a forest consisting of several trees or sub-trees). Every CA of a DApp has a single creator (parent) and can have several children that it can create (e.g., through the aforementioned patterns). Additionally, the CAs of the DApp can send each other message calls laterally without traversing the creation branches. Contracts of the DApp can also be called from outside of the DApp (see Fig. 3).

Message calls of CAs and EOAs: Message calls between accounts can be of different kinds, and are all documented in transaction traces with the mnemonic "CALL". (1) *Function calls* appear when an account calls a CA's function. Data attributes sent with the message call include, e.g., sender address, receiver address, input and output data, fees, and transfers of Ethereum's native token Ether (ETH) (which can be 0). (2) *Ether transfers without function calls*

[4] https://eips.ethereum.org/EIPS/eip-2535, accessed 2023-05-16

are solely directed to value transfers without additional input data and without triggering the execution of function logic. Accompanying data are, e.g., sender address, receiver address, fees, and the amount of transferred ETH.

Log entries: Log entries are used to communicate information about CAs' code execution to entities outside of the smart contract[5]. Log entries have to be specified in smart contract code in order to be emitted. Within limits, developers can choose what information shall be exposed with log entries. If smart contracts operate with ERC-tokens, however, developers are advised to implement standard interfaces including certain sets of events, e.g., to document token creation and transfers (e.g., ERC-20[6], ERC-721[7]).

Function call parameters as well as log entry parameters are emitted in encoded form and can be decoded using the corresponding Application Binary Interface (ABI) of a CA. ABIs are created at compile time and document a CA's interface functions (which can be called) as well as a set of log entries that are defined in the smart contract or inherited from other contracts[8]. An ABI's coverage of the log entries is not always complete. Before the recent introduction of Solidity v0.8.20, log entries according to the ERC-standard as well as log entries emitted through invoked code from imported libraries were not included in an ABI.

4 Data extraction method

The goal of the data extraction method is to gather as much data about a DApp as possible with minimum knowledge about the DApp. Therefore, the extraction method has to be capable of identifying DApp information from a small amount of input data. Hence, the extraction method takes as input a) *a non-empty set of accounts of a DApp*, and b) *a block range*. In order to extract the execution data of the DApp, two steps follow: 1) discover the creation sub-tree(s) of the DApp and 2) compute and transform execution data. Fig. 4 visualizes the extraction method that is described in this chapter. Since the approach exploits information of transaction traces and partially decodes them, it can be seen as an extension of [16].

1) Discover the creation sub-tree(s) of a DApp. We start with the *input set of accounts of the DApp* and *a block range*. All operations will take place within the specified block range. First, we retrieve transactions that have one of the input accounts as a sender or receiver account. We then look for transactions that contain message calls with ETH transfers from or to accounts from the input accounts. The newly identified transactions are replayed by an EVM of a Geth Ethereum Archival node to retrieve the full traces for the transactions. For that

[5] https://ethereum.org/en/developers/docs/smart-contracts/anatomy/#events-and-logs, accessed 2023-06-04
[6] https://eips.ethereum.org/EIPS/eip-20#events, accessed 2023-05-16
[7] https://eips.ethereum.org/EIPS/eip-721#specification, accessed 2023-05-16
[8] https://docs.soliditylang.org/en/v0.8.13/abi-spec.html, accessed 2023-05-16

purpose, a build-in debug tracer is used with the call tracer setting and instructed to output log entries[9]. We then forward and backward search for DApp accounts along the create sub-tree to identify: a) Child CAs: If the trace data includes contract creations by known DApp accounts these newly created CAs *(children)* are added to the set of DApp accounts. b) Parent accounts: If the trace data includes creations of smart contracts that are already known to be part of the DApp, the creating contracts *(parents)* are added to the set of DApp accounts. If any formerly unknown account was added to the set of DApp accounts, the previous three steps are repeated. The goal is to discover the whole create sub-tree of the DApp within the block range. If the input set contains accounts from different create sub-trees, several create sub-trees can be discovered.

That way, several types of information can be retrieved as raw data: 1) The creation sub-tree; meaning all parent or child contracts that belong to the specified set of DApp contracts within the block range. 2) All message calls that were executed in transactions that involved DApp contracts, including those with ETH transfers and function calls with encoded raw data within the block range. 3) Encoded raw data of all log entries that were emitted in a transaction that involved a DApp account in the specified block range.

2) Compute and transform execution data. The raw data of message calls and log entries are then decoded based on the Application Binary Interfaces[10] of DApp CAs that were identified in the transaction traces.

A CA's ABI specifies information to decode data of log entries and function calls in transaction traces. The ABI of a CA holds information about log entries that can be emitted by a smart contract with the exception of log entries according to an ERC-standard and log entries originating from imported libraries in CA code (as of Solidity v0.8.20 all log entries are included in the ABI, see Sec. 6). To maximize the number of retrieved log entries and to expand options for token tracking, ABIs of DApp CAs are expanded by adding standard events of ERC tokens[11]. Additionally, the ABI of a CA holds information about functions that can be called from outside the CA. All in all ABI specifications can be used to decode the log entry and function call raw data using existing libraries[12,13].

The extraction method makes use of the third-party service Etherscan[14] which provides access to Ethereum data. We used Etherscan to retrieve the transactions and ABIs of identified DApp CAs.

[9] https://geth.ethereum.org/docs/interacting-with-geth/rpc/ns-debug#debug_tracetransaction, accessed 2023-04-03
[10] https://ethereum.org/en/glossary/#abi, accessed 2023-06-03
[11] https://ethereum.org/en/developers/docs/standards/tokens/, accessed 2023-06-03
[12] https://github.com/iamdefinitelyahuman/eth-event/blob/master/eth_event/main-.py, accessed 2023-03-01
[13] https://web3py.readthedocs.io/en/latest/web3.contract.html#web3.contract.Contract.decode_function_input, accessed 2023-03-02
[14] https://etherscan.io/, accessed 2023-06-06

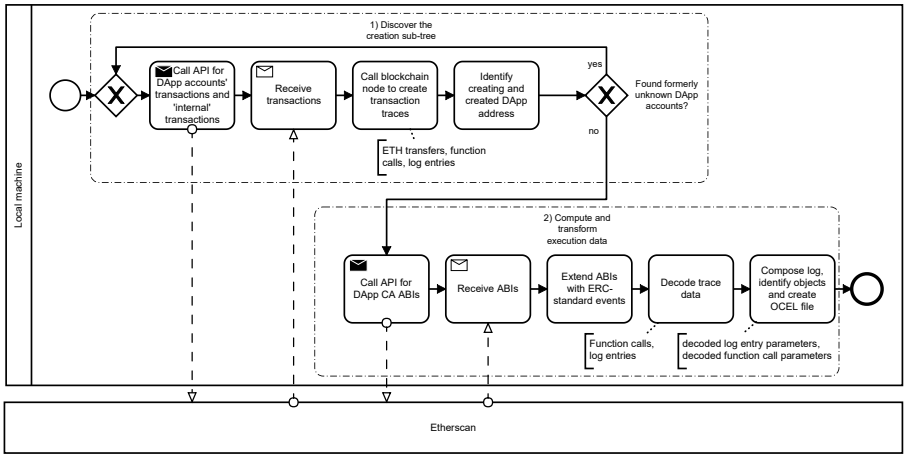


Fig. 4: Extraction method for creating object-centric event logs from blockchain data.

The extracted data is saved tabularly with one row representing a message call or a log entry. We use the PM4Py library function *convert_log_to_ocel()*[15] to convert the tabular data to OCEL. Note that while little domain knowledge is needed to extract the event log data, a certain level of domain knowledge is still needed to format the data to OCEL (e.g., for choosing data attributes as objects).

5 Data Analysis and Evaluation

For an initial validation of the approach, we conduct an assertion [22]. For the assertion, we choose the DApp Augur[16] (v1.0). Augur is an implementation of a betting platform on Ethereum. A bet starts by creating a *market* that contains a statement about a future event. Users can participate in the bet by placing assets in a market. If there is disagreement about the outcome of the event, users can create disputes. Once disputes are resolved, markets finalize and the bet is settled [17]. For Augur, a process mining case study based on a single-case notion event log was presented previously [9]. The previous case study will be used to compare the single notion approach to an object-centric approach in terms of data availability and possible insights. To retrieve the object-centric log, we employ the extraction method described in Sec. 4. We will use similar input data as [9]: we extract data starting from Augur's central logging CA[17] and for the blockrange [5926229, 11229573]. The resulting log is available for

[15] https://pm4py.fit.fraunhofer.de/static/assets/api/2.7.3/generated/pm4py.convert.convert_log_to_ocel.html, accessed 2023-07-18

[16] https://augur.net/, accessed 2023-06-04

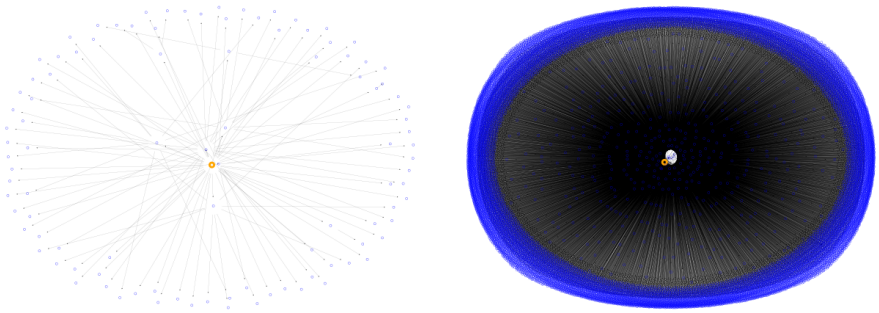
[17] 0x75228dce4d82566d93068a8d5d49435216551599

download[18]. In the following, we describe the resulting data (the creation tree and OCEL event log) and present a brief analysis of the data.

DApp contracts and creation sub-tree. The data extraction was heavily based on identifying CA creations during the observed block range. For Augur, we identified a total of 20866 CAs containing application logic and 1 EOA that served as an initial deployer. The high number of created CAs hints towards extensive use of the factory pattern in Augur's implementation. A visualization of the creation tree (as described in Sec. 3) is depicted in Fig. 5 - nodes represent accounts, and arcs represent creations. The majority of the DApp's CAs were created by a small number of contracts (nodes with many creations are plotted towards the inside of Fig. 5). To better show the tree structure, Fig. 5a contains a subset of 102 creations in which the arcs between creat*ing* accounts and creat*ed* accounts are clearly visible. Fig. 5b shows the entire creation tree. The accounts with the highest number of creations were given designated names on Etherscan. Those include the following accounts with the number of creations in parenthesis: *ShareTokenFactory* (7337)[19], *MapFactory* (3704)[20], *MarketFactory* (2944)[21], *InitialReporterFactory* (2917)[22], *MailboxFactory* (2909)[23], *DisputeCrowdsourcerFactory* (915)[24], *FeeTokenFactory* (142)[25], *FeeWindowFactory* (142)[26], *Augur Deployer* (40)[27], *UniverseFactory* (6)[28], and *ReputationTokenFactory* (2)[29].

Event log data. The resulting OCEL-log for Augur comprises 24 activities based on decoded log entries (934167) as well as 1 activity for message calls between accounts (117531). For the block range, all 11 activities of the XES-log from [9] also appear in the OCEL-log. The numbers of the activities' appearance match with one exception: 12 *contribute to dispute* events are missing in the OCEL-log (explanation in Sec. 6). Compared to the XES-log, the OCEL-log has 13 additional activities referring to (a) DApp administration, being *create universe* (1), *create fee window* (116), *sell complete sets* (863), *redeem fee window* (962), *create order* (24003), *fill order* (15194), and *cancel order* (15150), (b) activities describing token flow, being *burn token* (18456), *give approval* (54934), *mint tokens* (93593), *token was transferred* (284860), and *transfer token* (401854), and (c) activities referring to message and function calls with ETH

[18] https://ingo-weber.github.io/dapp-data/augur.html, accessed 2023-06-06
[19] 0x60a977354a6ba44310b2ee061bcf19632450e51d
[20] 0x67f53b749fe432274e3f53752a91da89ef86777e
[21] 0x518530aca60154403012f17c7b8e26f88f7494ee
[22] 0xbca52c29b535fd63bdc7ca35efa56116550f4c59
[23] 0xe33ca1ebb783343035b11a7e755c29c28b763540
[24] 0x1be98680ff697390cbc4cdc414a1be8add733bf7
[25] 0xe86a4beb10155a5bd7ebb430ce13438341e808a8
[26] 0x5b4140771615b25f22a4bf52f77e35cdccc5b663
[27] 0xd82369aaec27c7a749afdb4eb71add9e64154cd6
[28] 0xe62e470c8fba49aea4e87779d536c5923d01bb95
[29] 0x8fee0da3a35f612f88fb58d7028d14c7d99a3643



(a) Subset of 102 creations (b) 20 866 observed creations

Fig. 5: Creation tree of Augur (blue: DApp CAs; orange: EOA *Augur Deployer*).

transfers (117273), of which several are message calls into DApp accounts and which were decoded to retrieve human readable function names and input parameters (39043). The events in the log are described by a total of 95 attributes, from which we created three objects types: *DApp contracts*, *DApp users* and *markets*. We reckon that other object types including *transaction* or *block* might be beneficial for specialized types of analysis, too (e.g., for security analysis).

Analysis. For the analysis, we filtered the OCEL-log to include all common events with the XES-log, additional log entries from Augur's central logging CA, and token transfers. We included objects for *markets* and *DApp accounts*. The resulting object-centric directly-follows graph is depicted in Fig. 6. In addition to the results from [9] we can see that events occur that are not related to a *market* but are triggered by a DApp contract, e.g., *create universe*, *create fee window*, and *create order*. The event *transfer token* is only rarely triggered by the main logging contract. Instead a high number of other DApp accounts manage the token distribution in Augur.

As described in Sect. 2, a central advantage of object-centric logging formats is avoiding deficiency, convergence, and divergence that can happen when flattening data to suit a single case notion. We investigated if these advantages apply to the OCEL-log we extracted. In terms of *deficiency* we found activities that relate only to a selection of object types. E.g., the *market* notion does not cover the activities *create universe*, or *create fee window*. Additionally, token transfers between DApp accounts and user accounts cannot be related to markets either. The same is true for Ether transfers between accounts. In particular, when transfers and Gas fees are involved, deficiency should be avoided to cover process costs accurately. The Augur process had a total transfer volume of 349131 ETH between accounts during DApp execution (note that during one transaction ETH can be transferred from account to account and might count several times) which can go unnoticed focusing on the market case notion only. *Convergence* would appear in the Augur log mainly for activities relating to the

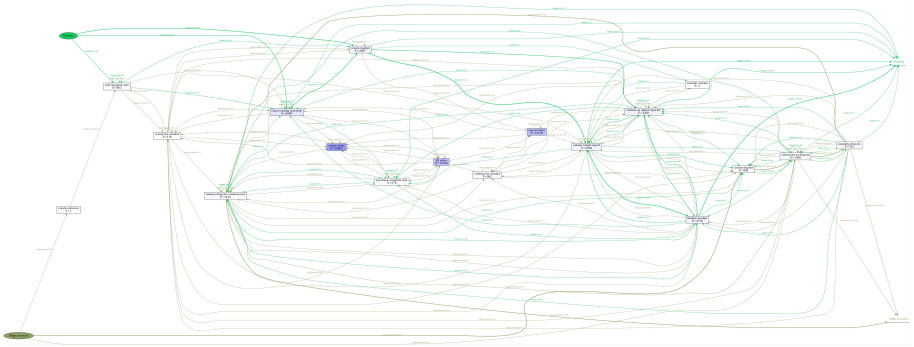


Fig. 6: Object-centric directly-follows graph discovered with contracts, markets, and transactions as objects.

market as well as the user account notion. Those include, e.g, *create market*, *redeem as initial reporter*, *create dispute*, etc., all of which would have to be duplicated to represent the process in two event logs. *Divergence* becomes an issue depending on the chosen notion. Events could appear related through a common CA object although they are not. To illustrate the divergence issue with Augur as an example: Augur is one DApp consisting of multiple CAs. The code in the CAs execute different parts of the application. E.g., market behavior is processed in CA 0xA. In parallel an incentive system with a native token is implemented in CA 0xB. If the single case notion *DApp account* was selected, different token transfers between different users that were active in different markets might appear related, although they are not.

6 Discussion

The analysis in the previous section illustrated benefits of using an object-centric logging format over a single case notion format. Applying our approach and the subsequent analysis also uncovered several points to debate.

The current implementation of this paper's data extraction method relies on the third-party service Etherscan. The source code of Etherscan is not fully public so its use for data retrieval reduces transparency in the research process. However, that dependence was accepted for two reasons: 1) *Etherscan is an indexed data store of the Ethereum blockchain.* To reduce the size of the Ethereum blockchain, only the results of transactions are stored on-chain, while the operations leading up to the transactions' outcomes (transaction traces) are not. For querying data from a blockchain archival node that means: for a CA 0xA, only those transactions can be queried that have 0xA as a sender or receiver of a transaction in a certain block range. It is also possible that CA 0xA received a message call during the execution of a transaction from the sender EOA 0xB to the receiver CA 0xC. In that case, the message call 0xC $\rightarrow$ 0xA is not stored on-chain. Indexing services such as Etherscan provide data stores that save (a

subset of) such message calls and make them easily queriable. It is also possible to circumvent using a third-party service by creating a local database containing message calls between all active accounts within a block range. 2) *Retrieving DApp CA's ABIs.* Log entry data as well as function names and inputs are stored on the blockchain encoded. Decoding the data requires information from the CA's ABI. Etherscan provides a number of verified ABIs for CAs, which we drew from. If the smart contract code of the CA is known, the ABI can also be computed by a Solidity compiler.

To discover all accounts belonging to a DApp, the extraction method hinges on knowing at least one contract of each creation sub-tree. If multiple root deployers exist, creation sub-trees may partially not be discovered. Note, however, that transaction traces of undiscovered sub-tree accounts can still be logged, if they contain message calls or creations by known DApp accounts. That was also the case in the Augur log. 12 *contribute to dispute* events were missing in our extracted data. We investigated the issue and found a second deployer EOA that is not labeled on Etherscan[30]. As a result, a fraction of the DApps transactions could not be discovered. Vice versa, if a root deployer account was used to deploy more than a single DApp, the extraction method could discover accounts that do not belong to the DApp and mistakenly extract non-DApp-account data. Precautions can be taken by choosing distinguished block ranges during extraction or explicitly excluding accounts from considered candidates of DApp accounts. Both countermeasures require a level of domain knowledge though.

Furthermore, the extraction method assumes transparency about log entries of a CA. Types of log entries are documented in a CA's ABI. In Solidity, ABIs only include information about log entries that are explicitly included in a CA's smart contract code and inherited log entries of another smart contract[31]. There may, however, be log entries emitted by library code of a CA, which were not included in the ABI. This paper's extraction method thus, could not capture library log entries for Augur. Library log entries were only recently included in the ABI for new Solidity compiler versions[32] (as of Solidity v0.8.20[33]). For smart contracts written in older Solidity code, however, the issue persists.

This paper's approach was implemented with the Geth-based Ethereum client Erigon, so that the rich features of Geth were available for transaction trace generation. We cannot make a statement about reproducibility with other clients except Geth and Erigon. However, both clients have a combined market share of ca. 70% of all Ethereum clients, which makes the approach accessible for the

[30] 0x57f1c2953630056aaadb9bfbda05369e6af7872b
[31] https://github.com/ethereum/solidity/issues/13086, accessed 2023-06-03
[32] https://github.com/ethereum/solidity/pull/10996, accessed 2023-06-03
[33] https://github.com/ethereum/solidity/releases/tag/v0.8.20, accessed 2023-06-06

majority of node operators[34].

We also noticed limitations when using OCEL. In blockchain applications such as Augur, different types of objects exist, e.g., DApp CAs, transactions, user accounts, etc. Some of these objects might change their role as a process instance progresses. E.g., the same user account in Augur can be sender as well as receiver of a token. Documenting an object's changing roles over time is currently not explicitly supported in OCEL. Hence an integrated picture showing full traces of an object with its changing roles cannot be depicted. The same phenomenon was described by [12, 15] as an issue in representing "object evolution".

Another challenge for log generation is determining the order of events in a blockchain environment. In past studies, a blockchain event's timestamp was defined as the timestamp of the block it was included in. There may, however, be an order to events or transactions within a block. Similarly, EVM traces of individual transactions represent a (tree) structure with an order of events. Events and message calls in the same block appear to have occurred at the same time when only considering the inclusion block's timestamp. OCEL requires a timestamp to determine the order of events and does not allow additional complementary ordinal variables. We hence had to manipulate the timestamps to preserve the events' and message calls' order.

The presentation of the paper's approach is heavily based on the technology of Ethereum blockchains. In addition to the Ethereum Mainnet, from which we extracted the data, the approach is also applicable on other Ethereum-based networks used in enterprises. Some of the concepts we made use of are transferable to other blockchains. For one, executable code is deployed as smart contracts also in other second generation blockchains (e.g., Hyperledger Fabric[35] and can be grouped to identify code belonging to one application. Also the notion of transactions of assets is a concept shared by all blockchains [21, p.5] and can be exploited to retrieve object-centric process mining logs across platforms.

7 Conclusion

Based on the literature, we showed shortcomings of the single notion logging format XES to capture execution data of DApps. We presented an approach to retrieving execution data from dynamically deployed blockchain applications with little prior knowledge about the application. Based on one case application, we examined the suitability of the current object-centric logging standard in process mining as an event log format for blockchain applications' execution data. For the considered case, we observe a first indication that OCEL is a suitable format with respect to mapping the distributed nature of blockchain logging in different accounts and varying levels of object types (CAs, EOCs, tokens,

[34] https://clientdiversity.org/#distribution, accessed 2023-06-03

[35] https://hyperledger-fabric.readthedocs.io/en/latest/smartcontract/smartcontract.html, accessed 2023-07-18

transactions, etc.) and tackles deficiency, convergence, and divergence issues. We also note that OCEL appears to be unsuitable for depicting the transition of objects between roles within the same activity, e.g., an EOC being a sender in one event *Transfer* and a receiver in another event *Transfer*.

References

1. van der Aalst, W.M.P.: Process Mining: A 360 Degree Overview, pp. 3–34. Springer International Publishing, Cham (2022)
2. Van der Aalst, W.M., Günther, C., Bose, J., Carmona Vargas, J., Dumas, M., van Geffen, F., Goel, S., Guzzo, A., Khalaf, R., Kuhn, R., et al.: Ieee standard for extensible event stream (xes) for achieving interoperability in event logs and event streams. IEEE Std 1849-2016 pp. 1–50 (2016)
3. Aalst, W.: Object-Centric Process Mining: Dealing With Divergence and Convergence in Event Data. In: Ölveczky, P., Salaün, G. (eds.) Software Engineering and Formal Methods (SEFM 2019). Lecture Notes in Computer Science, vol. 11724, pp. 3–25. Springer-Verlag, Berlin (2019)
4. Bandara, H.D., Bockrath, H., Hobeck, R., Klinkmüller, C., Pufahl, L., Rebesky, M., van der Aalst, W., Weber, I.: Event logs of ethereum-based applications (2021)
5. Beck, P., Bockrath, H., Knoche, T., Digtiar, M., Petrich, T., Romanchenko, D., Hobeck, R., Pufahl, L., Klinkmüller, C., Weber, I.: Blf: A blockchain logging framework for mining blockchain data. In: BPM (PhD/Demos). pp. 111–115 (2021)
6. Corradini, F., Marcantoni, F., Morichetta, A., Polini, A., Re, B., Sampaolo, M.: Enabling Auditing of Smart Contracts Through Process Mining, pp. 467–480. Springer International Publishing, Cham (2019)
7. Ghahfarokhi, A.F., Park, G., Berti, A., van der Aalst, W.M.: Ocel: A standard for object-centric event logs. In: New Trends in Database and Information Systems: ADBIS 2021 Short Papers, Doctoral Consortium and Workshops: DOING, SIMPDA, MADEISD, MegaData, CAoNS, Tartu, Estonia, August 24-26, 2021, Proceedings. pp. 169–175. Springer (2021)
8. Günther, C.: XES Standard Definition. www.xes-standard.org (2009)
9. Hobeck, R., Klinkmüller, C., Bandara, H.D., Weber, I., van der Aalst, W.: Process mining on blockchain data: A case study of Augur. In: BPM'21: International Conference on Business Process Management. pp. 306–323. Rome, Italy (Sep 2021)
10. IEEE Task Force on Process Mining: Process Mining Manifesto. In: BPM Workshops. Lecture Notes in Business Information Processing, vol. 99. Springer-Verlag, Berlin (2011)
11. Klinkmüller, C., Ponomarev, A., Tran, A.B., Weber, I., van der Aalst, W.M.P.: Mining blockchain processes: Extracting process mining data from blockchain applications. In: BPM Blockchain Forum. pp. 71–86 (2019)
12. M'Baba, L.M., Assy, N., Sellami, M., Gaaloul, W., Nanne, M.F.: Extracting artifact-centric event logs from blockchain applications. In: 2022 IEEE International Conference on Services Computing (SCC). pp. 274–283. IEEE (2022)
13. M'Baba, L.M., Sellami, M., Gaaloul, W., Nanne, M.F.: Blockchain logging for process mining: a systematic review. In: HICSS 2022: 55th Hawaii International Conference on System Sciences. pp. 6197–6206. HICSS (2022)
14. Mendling, J., Weber, I., Aalst, W., Brocke, J., Cabanillas, C., Daniel, F., Debois, S., Ciccio, C.D., Dumas, M., Dustdar, S., Gal, A., Garcia-Banuelos, L., Governatori, G., Hull, R., Rosa, M.L., Leopold, H., Leymann, F., Recker, J., Reichert, M.,

Reijers, H., Rinderle-Ma, S., Solti, A., Rosemann, M., Schulte, S., Singh, M., Slaats, T., Staples, M., Weber, B., Weidlich, M., Weske, M., Xu, X., Zhu, L.: Blockchains for Business Process Management: Challenges and Opportunities. ACM Transactions on Management Information Systems **9**(1), 1–16 (2018)
15. Moctar M'Baba, L., Assy, N., Sellami, M., Gaaloul, W., Farouk Nanne, M.: Process mining for artifact-centric blockchain applications. Simulation Modelling Practice and Theory **127**, 102779 (2023). https://doi.org/https://doi.org/10.1016/j.simpat.2023.102779, https://www.sciencedirect.com/science/article/pii/S1569190X23000564
16. Mühlberger, R., Bachhofner, S., Di Ciccio, C., García-Bañuelos, L., López-Pintado, O.: Extracting event logs for process mining from data stored on the blockchain. In: Business Process Management Workshops. pp. 690–703 (2019)
17. Peterson, J., Krug, J., Zoltu, M., Williams, A.K., Alexander, S.: Augur: A decentralized oracle and prediction market platform. Tech. rep., Forecast Foundation (July 12, 2018), https://github.com/AugurProject/whitepaper/blob/master/v1/english/whitepaper.pdf, accessed 2021-01-05
18. Wirawan, N.Y., Yahya, B.N., Bae, H.: Incorporating Transaction Lifecycle Information in Blockchain Process Discovery, pp. 155–172. Springer Singapore, Singapore (2021)
19. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**(2014), 1–32 (2014)
20. Xu, X., Pautasso, C., Zhu, L., Lu, Q., Weber, I.: A pattern collection for blockchain-based applications. In: Proceedings of the 23rd European Conference on Pattern Languages of Programs. EuroPLoP '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3282308.3282312, https://doi.org/10.1145/3282308.3282312
21. Xu, X., Weber, I., Staples, M.: Architecture for Blockchain Applications. Springer (2019)
22. Zelkowitz, M., Wallace, D.: Experimental models for validating technology. Computer **31**(5), 23–31 (1998). https://doi.org/10.1109/2.675630