University of Camerino

School of Science and Technology

Master's Degree in Computer Science (LM-18)

Application of Process Mining to blockchain-based applications

Student
Yuri Paoloni

Supervisor
Andrea Morichetta

Co-Supervisor
Barbara Re

A.Y. 2021/2022

Abstract

Through smart contracts, blockchain has become a technology for managing trustless peer-to-peer exchanges of digital assets, paving the way for new forms of trade and business. In such a scenario, the application of Process Mining can help understand processes and their actual execution in ways other strategies cannot. However, due to the structure of blockchain data, applying Process Mining in such a context is challenging. The techniques created so far by researchers have limitations when applied to smart contracts. Some are tailored to specific use cases like blockchain-based Business Process Management Systems. Others require the data under analysis to be in a precise format which is uncommon for a significant portion of the existing smart contracts. To solve this challenge, we propose an application-agnostic extraction methodology to collect data from every EVM-compatible smart contract and enable the application of Process Mining techniques. The proposed methodology focuses on blockchain transactions and their internal execution. The former, unlike events, have standard parameters that ensure a common ground of operations. The latter represents the execution flow of transactions, which is the invocations of the functions of smart contracts involved in the process. The methodology comprises five steps: (i) extraction of data from smart contracts, (ii) cleaning of raw data, (iii) selecting and defining sorting criteria, (iv) trace construction, and, finally, (v) XES log generation. An in-depth case study of Decentraland, a metaverse and digital assets marketplace developed on the Ethereum blockchain, has been carried out to demonstrate the validity of the proposed methodology. We were able to generate XES logs from different block ranges and one or more Decentraland smart contracts combined. Unlike event data collection, a plain execution with transactions does not require prior knowledge of smart contract code. On the generated XES logs, we successfully applied several Process Mining techniques like Simulation, Social Network Analysis, and Process discovery.

Contents

Listings

List of Figures

List of Tables

1. Introduction

In recent years, blockchain has been acquiring attention as a technology to execute business processes and regulate the interactions between the involved parties. The use of smart contracts makes it possible to execute complex processes (i.e., involving many business parties) in a trustless environment without the need to rely on a trusted party. Such a scenario paved the way for new forms of trade and business: exchange of digital assets, track of physical goods, complex business interactions, and more. Process Mining techniques have been vastly employed on traditional systems to exploit and deeply understand business processes. Examples are: compare a process's actual execution with the intended one (i.e., conformance checking); produce high-level models from process logs (i.e., process discovery); analyze business parties' interactions (i.e., social network analysis). In theory, it should be possible to achieve the same results on business processes executed on the blockchain. However, due to the structure of blockchain data, applying Process Mining in such a context is challenging. Blockchain data is not natively suitable for Process Mining: (i) the notion of "trace" is not present (i.e., there isn't a process identifier that groups a set of transactions together); (ii) timestamps are not accurate because transactions mined in the same block have the same timestamp. In recent years, researchers have proposed multiple techniques to ease this task. Yet, there are limitations in the application to smart contracts structurally different from the intended target. For instance, [6] is tailored to blockchain-based Business Process Management Systems. This kind of system is generated from a BPMN model. Hence, we know a priori the context, the activities, and the execution flow of the process under analysis. These assumptions do not hold for every smart contract in the blockchain. Other techniques, like [4] and [2], focus on events. Events are logs issued during the execution of smart contract functions. Events don't have standard parameters and are not mandatorily present in a function body. To enable Process Mining, they must contain a shared parameter (i.e., case ID) and be emitted correctly and in the right place.

To solve this challenge, we propose an application-agnostic extraction methodology to collect data from every smart contract and enable the application of Process Mining techniques. The proposed methodology focuses on blockchain transactions and their internal execution. The transactions, unlike events, have standard parameters that ensure a common ground of operations. In every transaction, we can rely on the fact that the *hash*, the *sender*, the *receiver*, and other parameters are present. The transactions' internal execution represents the execution flow of transactions, which is the invocations of the functions of the smart contracts involved in the procedure. The methodology comprises five steps:

- *Data extraction*: the user selects the smart contracts and, optionally, the block range for the data retrieval.

- *Raw data cleaning*: the transaction parameters are converted into a readable format since some are hexadecimal. Moreover, unnecessary parameters are removed.
- *Select and define sorting criteria*: as mentioned above, timestamps of transactions in the same block are equal. We need additional or different criteria to sort transactions.
- *Traces construction*: a parameter is selected as case ID to construct traces.
- *XES log generation*

An in-depth case study of Decentraland, a metaverse and digital assets marketplace developed on the Ethereum blockchain, has been carried out to demonstrate the validity of the proposed methodology. We chose Decentraland because Process Mining has not been applied extensively to metaverses and the variety of functions it offers. (e.g., cryptocurrencies and NFTs exchange, virtual lands, games, and more). We were able to generate XES logs from different block ranges and one or more Decentraland smart contracts combined. On the generated XES logs, we successfully applied and evaluated several Process Mining techniques like Process Discovery, Simulation, and Social Network Analysis. Special attention was given to the logs generated from the transactions' internal execution on which we applied BPMN Miner [1], a novel technique able to output a BPMN model with subprocesses inside.

In the following, we first introduce relevant background information on the blockchain, Ethereum, process mining, and XES logs in Section 2. The related work is described in Section 3. The proposed methodology is introduced in Section 4, implemented in Section 5, and evaluated in Section 6. Finally, Section 7 concludes and lists future challenges.

2. Background

2.1 Blockchain

A **blockchain** is a growing list of records, called blocks, that are securely linked together using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. As each block contains information about the block previous to it, they form a chain, with each additional block reinforcing the ones before it. Therefore, blockchains are resistant to modification of their data because once recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks. Blockchains are typically managed by a peer-to-peer network where nodes collectively adhere to a protocol to communicate and validate new blocks.

2.1.1 Ethereum

Ethereum is a decentralized, open-source blockchain with smart contract functionality. Ether (ETH) is the cryptocurrency generated by the Ethereum protocol as a reward to miners in a proof-of-work system for adding blocks to the blockchain. Ethereum was the first blockchain to include computational capabilities with smart contracts. Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met. They are typically used to automate the execution of an agreement so that all the participants can be immediately certain of the outcome, without an intermediary's involvement or time loss. They can also automate a workflow by triggering the next action when specific conditions are met. Smart contracts are written with the Solidity programming language and executed on the Ethereum Virtual Machine (EVM), similar to common programming languages. Ethereum is the main blockchain for smart contract development.

2.1.2 Transactions

Transactions are cryptographically signed instructions from accounts sent on the blockchain. An account will initiate a transaction to update the state of the blockchain network. The simplest transaction is the transferring of cryptocurrencies (i.e., ETH) from one account to another. A more complex example of a transaction is the execution of a smart contract function. The transactions which involve smart contracts are executed by miners through the Ethereum Virtual Machine (EVM), which, similarly to the Java Virtual Machine, converts the transaction instructions (i.e., Solidity code) into bytecode to execute them. Once the execution is completed, the transaction is added to a block and propagated in the network. In the Ethereum blockchain, a submitted transaction includes the following information:

- hash - transaction hash
- block number - the number of the block in which the transaction was included
- timestamp: time at which the transaction has been added in a block
- to – the receiving address (if an externally-owned account, the transaction will transfer value. If a contract account, the transaction will execute the contract code)
- from – the identifier of the sender. This is generated when the sender's private key signs the transaction and confirms the sender has authorized this transaction
- value – amount of ETH to transfer from sender to recipient (in WEI, a denomination of ETH)
- data – optional field to include arbitrary data (e.g., description of transaction, binary code to create smart contract, function invocation)
- gasLimit – the maximum amount of gas units that can be consumed by the transaction. Units of gas represent computational steps
- maxPriorityFeePerGas - the maximum amount of gas to be included as a tip to the miner
- maxFeePerGas - the maximum amount of gas willing to be paid for the transaction (inclusive of baseFeePerGas and maxPriorityFeePerGas)

2.1.3 Transactions internal execution

A **transaction's internal execution**, also called internal transaction, is the consequence of smart contract logic triggered by an external transaction. A smart contract engagement can result in tens or hundreds of internal transactions. These represent value transfers that occur when a smart contract or a token transfer is executed. Internal transactions, unlike regular transactions, lack a cryptographic signature and are typically stored off-chain, meaning they are not a part of the blockchain itself. Transaction's internal executions can be retrieved by recording all the value transfers that took place as part of external transaction execution. The following is an example of the data obtainable for transactions internal execution using EthTx[1] which is an advanced decoder for blockchain transaction developed by TokenFlow[2].


```
{
    "LOAD_ID": "2022-05-09 14:57:50.000",
    "CHAIN_ID": "mainnet",
    "BLOCK": 12010461,
    "TIMESTAMP": "2021-03-10 11:14:20.000",
    "TX_HASH": "0
        x15ee97483fdada5e3ec49991f4de338e554c18b44a59fb3612d84faf99b52dcd
        ",
    "CALL_ID": null,
```


[1] https://ethtx.info/
[2] https://tokenflow.live/

```
    "CALL_TYPE": "call",
    "FROM_ADDRESS": "0x28cd504f564c7288413148a78788e29e7fffebf0
        ",
    "FROM_NAME": "0x28cd504f564c7288413148a78788e29e7fffebf0",
    "TO_ADDRESS": "0xf87e31492faf9a91b02ee0deaad50d51d56d5d4d",
    "TO_NAME": "LAND",
    "FUNCTION_SIGNATURE": 22465,
    "FUNCTION_NAME": "setApprovalForAll",
    "VALUE": 0,
    "ARGUMENTS": "{\"address\":\"0
        x8e5660b4ab70168b5a6feea0e0315cb49c8cd539\",\"approved
        \":\"True\"}",
    "RAW_ARGUMENTS": "[{\"name\":\"address\",\"raw\":\"0
        x0000000000000000000000008e5660b4ab70168b5a6feea0e0315cb49c8cd539
        \",\"type\":\"address\"},{\"name\":\"approved\",\"raw
        \":\"0
        x0000000000000000000000000000000000000000000000000000000000000001
        \",\"type\":\"bool\"}]",
    "OUTPUTS": "{}",
    "RAW_OUTPUTS": "[]",
    "GAS_USED": 29105,
    "ERROR": null,
    "STATUS": true,
    "ORDER_INDEX": 1315,
    "DECODING_STATUS": true,
    "STORAGE_ADDRESS": "0
        xf87e31492faf9a91b02ee0deaad50d51d56d5d4d"
}
```


The *CALL_ID* field allows for distinguishing a transaction from the internal transaction execution. When it is *null* it indicates that the record is a transaction. Instead, for subcalls, it can assume different values indicating deeper levels in the trace. Example are: *0*, *0_0* (subcall of 0), *0_1* (on the same level of *0_0*), *0_0_0*, *0_2*, *0_2_0*, etc...

Putting together the internal transactions, we can get the full execution trace of a transaction. Such data can be processed using Process Mining to get details about process variants. For instance, an infrequent variant could indicate a bug in the immutable code.

2.1.4 Events

Events are logs issued during the execution of smart contract functions. An event hasn't standard parameters and is not mandatorily present in a function body. The smart contracts developers add the events where they need to log something out. In general, events log the outcome of an operation (e.g., transfer of a token, deposit). Logged data is used by external services, like frontends, to update their internal states accordingly. The following is a simple code example that shows the use of events to log that the value has changed.

```
pragma solidity 0.5.17;
```

```
contract Counter {

    event ValueChanged(uint oldValue, uint256 newValue);

    // Private variable of type unsigned int to keep the
       number of counts
    uint256 private count = 0;

    // Function that increments our counter
    function increment() public {
        count += 1;
        emit ValueChanged(count - 1, count);
    }

    // Getter to get the count value
    function getCount() public view returns (uint256) {
        return count;
    }

}
```

2.2 Process Mining

Process mining is a family of techniques relating to the fields of data science and process management to support the analysis of operational processes based on event logs. The goal of process mining is to turn event data into insights and actions. Process mining is an integral part of data science, fueled by the availability of event data and the desire to improve processes. Process mining techniques use event data to show what people, machines, and organizations are doing. Process mining provides novel insights that can be used to identify the execution path taken by operational processes and address their performance and compliance problems.

Process mining starts from event data. Input for process mining is an event log. An event log views a process from a particular angle. Each event in the log should contain (i) a unique identifier for a particular process instance (called case id), (ii) an activity (description of the event that is occurring), and (iii) a timestamp. There may be additional event attributes referring to resources, costs, etc., but these are optional. With some effort, such data can be extracted from any information system supporting operational processes. Process mining uses these event data to answer a variety of process-related questions. There are three main classes of process mining techniques: process discovery, conformance checking, and process enhancement.

2.2.1 Process Discovery

2.2.2 Any other techniques

Add a subsection for each applied technique

2.3 XES logs

The XES [3] standard is officially published by IEEE. The XES standard defines a grammar for a tag-based language to provide a unified and extensible methodology for capturing event logs and event streams. An XML Schema for event log and extensions is part of the standard. A XES instance corresponds to a file-based event log that can be used to transfer event-driven data from a site generating this event-driven data to a site where this data will be analyzed. The primary purpose of XES is for Process Mining.

A XES file is structured as follows. On the top level there is one **log** object which contains all event information related to one specific process. A log contains an arbitrary number of **trace** objects. Each trace describes the execution of one specific instance, or case, of the logged process. Every trace contains an arbitrary number of event **objects**. Events represent atomic granules of activity that have been observed during the execution of a process. The log, trace, and event objects contain no information themselves. They only define the structure of the document. All information in an event log is stored in attributes. All attributes have a string-based key.

```
<log xes.version="1.0" xes.features="nested-attributes"
    openxes.version="1.0RC7" xmlns="http://www.xes-standard.org
    /">
        <string key="concept:name" value="TEST"/>
        <string key="lifecycle:model" value="standard"/>
        <trace>
                <string key="concept:name" value="1411"/>
                <event>
                        <string key="id" value="1411"/>
                        <string key="org:resource" value="user
                            "/>
                        <date key="time:timestamp" value
                            ="1970-03-27T12:02:00+10:00"/>
                        <string key="lifecycle:transition"
                            value="complete"/>
                        <string key="concept:name" value="A"/>
                </event>
                <event>
                        <string key="id" value="1411"/>
                        <string key="org:resource" value="user
                            "/>
                        <date key="time:timestamp" value
                            ="1970-03-27T12:32:00+10:00"/>
                        <string key="lifecycle:transition"
                            value="complete"/>
                        <string key="concept:name" value="S1A
                            "/>
                        <string key="ids1" value="2S2505"/>
                </event>
        </trace>
</log>
```

3. Related work

Before starting my research on Process Mining and blockchain, I decided to examine the existing work on the topic. I searched on Google Scholar using the text parameter *"process mining blockchain"*, and I found out that in recent years, researchers have proposed multiple methodologies to apply Process Mining to blockchain data. To skim the obtained results, I further analyzed only the papers that respected the following criteria:

- Implement an extraction methodology
- Deal with the Ethereum blockchain
- Considerable number of citations (i.e., >100)

After the filtering, the remaining papers were: *Extracting Event Logs for Process Mining from Data Stored on the Blockchain*[6], *Mining Blockchain Processes: Extracting Process Mining Data from Blockchain Applications*[4], and *Process Mining on Blockchain Data: A Case Study of Augur*[2]. The examination of the mentioned works highlighted limitations and room for additional research. The following will describe the three papers and their limitations.

3.1 Process Mining on Blockchain-based BPMS

Di Ciccio et al. in [6] defined a methodology to extract logs from the transactions involving smart contracts generated with a blockchain-based Business Process Management System (BPMS) like Caterpillar[5] and Lorikeet[7]. This kind of system generates smart contracts code from BPMN or Choreography models. Hence we know a priori: (i) the context and the scenario where the operations take place, (ii) the set of allowed activities, (iii) and the execution flow of the system. These assumptions do not apply to most of the smart contracts deployed in the Ethereum blockchain because, even if they were present, we would not have access to the models underlying the smart contract logic. The proposed methodology starts from an activity dictionary containing the activity names, function signatures, and other information. An example of the dictionary is shown in figure 3.1. Traces are built using the *transaction.to* field. Hence, every process instance refers to the transactions with a specific smart contract as the receiver.

The authors successfully evaluated the methodology on an Incident-management process [8]. It is evident that the methodology has huge limits in terms of flexibility and applies only to the intended target (i.e., blockchain-based BPMS). However, this limitation is intended by the authors, who decided to focus specifically on blockchain-based BPMS.

Activity name	Function signature	Function selector
Customer has a problem	`Customer_Has_a_Problem()`	`0xefe73dcb`
Get problem description	`Get_problem_description(int32 x)`	`0x92ed10ef`
Ask 1st level support	`Ask_1st_level_support(int32 y)`	`0x82b06df7`
Explain solution	`Explain_solution()`	`0x95c07f19`
Ask 2nd level support	`Ask_2nd_level_support()`	`0x63ad6b81`
Provide feedback for account manager	`Provide_feedback_for_account_manager()`	`0x58a66413`
Ask developer	`Ask_developer()`	`0xecb07b8c`
Provide feedback for 1st level support	`Provide_feedback_for_1st_level_support()`	`0x3b26a0ea`
Provide feedback for 2nd level support	`Provide_feedback_for_2nd_level_support()`	`0x9ec3200a`

Figure 3.1: Activity dictionary

3.2 Ethereum Logging Framework (ELF)

The other analyzed papers are *Mining Blockchain Processes: Extracting Process Mining Data from Blockchain Applications*[4] and the respective use case[2]. Weber et al. proposed a framework to extract process event data from DApps that utilize the Ethereum's transaction log as storage for logged data (i.e., Solidity events[1]). The framework includes (i) a *manifest* to specify transformation rules (i.e., Ethereum logs → XES data), (ii) an extractor that applies the manifest rules, and (iii) a *generator* that produces Solidity code to emit events in line with the manifest specifications. To extract data, the user must create a manifest containing the smart contract addresses, the block range, and the events he wants to analyze. Additionally, he needs to specify mappings from the events data to XES data. This configuration step requires knowledge of the target smart contracts code and adds complexity to the data extraction process. As described in Section 2, events haven't standard parameters and are not mandatorily present in a function body. Nonetheless, the framework relies on the correct issuance of events. ELF needs smart contracts to extensively emit events with enough data to create a correlation between them. Each event must contain a shared attribute key to build traces and must be emitted in the functions that we're interested in. For instance, if a function lacks event emission, the framework will generate an XES log without that function and potentially lead to wrong results (e.g., process discovery model lacking an activity). This restricts the application of ELF to well-written smart contracts and compliant events. To partially address these issues, the authors provided a *generator* that produces Solidity code to emit events correctly. However, this will work mainly for new smart contracts. It is unlikely that a team decides to deploy again an existing smart contract to make it compliant with the Ethereum Logging Framework. The deployment of a new smart contract on Ethereum costs thousands of dollars and requires all the services relying on the smart contract to be updated with the new address.

ELF was successfully evaluated on CryptoKitties[2] and Augur[3][2]. The evaluation on Augur is particularly interesting, as it identified a bug, albeit not a critical one, in the smart contract. This proves that the framework is valid but has limitations in the application to every smart contract.

The table 3.1 presents an overview of the analyzed papers with the *methodology*, the *target data*, the *target application*, *case ID*, and the *process discovery algorithm* since

[1] `https://docs.soliditylang.org/en/v0.8.16/contracts.html#events`
[2] `https://www.cryptokitties.co/`
[3] `https://augur.net/`

it was applied in all the three papers.

Methodology	**Data**	**Target**	**Case ID**	**Discovery algorithm**
ELF[4]	Event	CryptoKitties	Kitty ID	DFGs
ELF[2]	Event	Augur	Market ID	Inductive Miner
BPMS[6]	Transaction	Incident-management[8]	transaction.to	Inductive Visual Miner

Table 3.1: Related work overview

4. Methodology

The analysis of the related work in Section 3 shown that the present research is missing an application-agnostic methodology to enable the application of Process Mining techniques on blockchains easily. A methodology able to collect data from every smart contract and generate XES logs ready for Process Mining without requiring complex configurations. To work with every smart contract, the methodology needs data with standard parameters, which, unlike events, gives a common ground for operations. Blockchain transactions, as described in Section 2, have standard parameters and, for this reason, have been selected as the input data of the methodology. In addition to the transaction, the methodology will focus on transaction internal execution in order to exploit the execution trace of smart contract functions through Process Mining. Transaction internal execution have been as well described in Section 2.

The proposed methodology, as depicted in the figure 4.1, consists of five steps (i.e., *data extraction*, *raw data cleaning*, *sorting*, *trace construction*, and *XES generation*) that are described in the following sections.

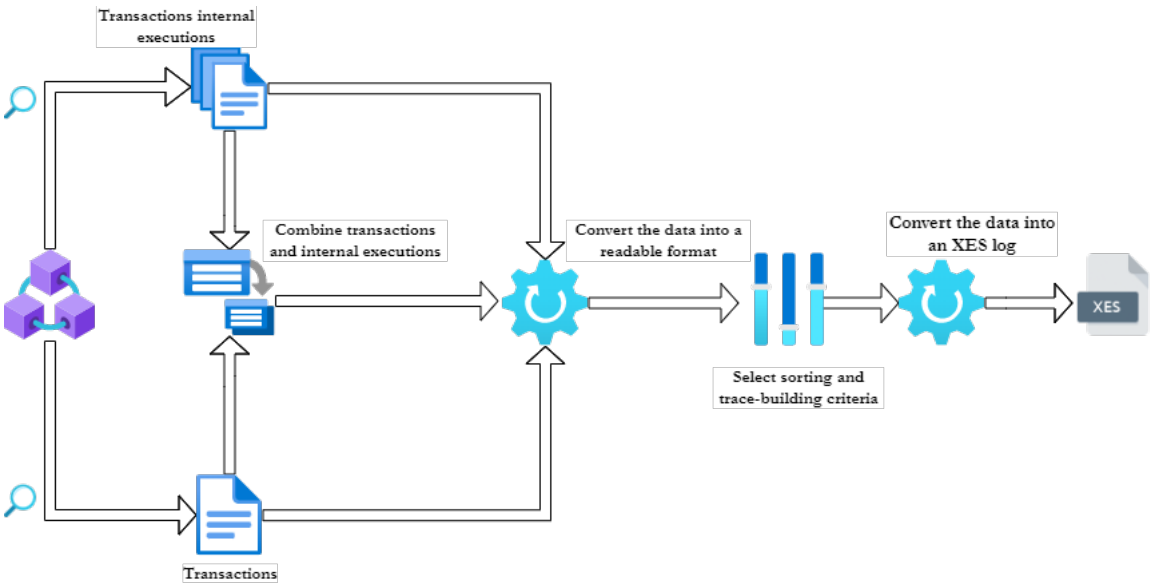

Figure 4.1: Overview of the methodology

4.1 Data extraction

The *data extraction* step is the first step of the methodology. As the name implies, blockchain data is extracted according to two values provided by the user: smart con-

tract addresses and block range. The former indicates the addresses we want to collect data about. The latter is optional and is used to scope the research in a precise time frame. The blockchain data that the methodology is going to collect are transactions and the respective internal execution. The reasons behind the usage of transactions as input data have been explained in the previous sections.

4.1.1 Transactions

The collected transactions are in the format described in Section 2. Transactions are collected using a blockchain explorers like Etherscan [1].

Algorithm 1 Transactions retrieval

Require: $contractsAddresses, startBlock, endBlock$

1: **for all** *contract* in $contractsAddresses$ **do**
2: $transactions = etherscan.getTxsByAddress(contract, startBlock, endBlock)$
3: **end for**

4.1.2 Transactions internal execution

Internal transactions are not a part of the blockchain itself. They are typically stored off-chain by recording the value transfers that took place during a transaction. To collect such data, EthTx[2] can be used. It is an advanced decoder for blockchain transactions which returns a lot of information about a transactions including emitted events, tokens transfers, and the trace of internal transactions. The algorithm to collect transactions internal execution is similar to Algorithm 1 with the only difference of EthTx instead of Etherscan. The data obtained is in the format shown in Section 2.

4.2 Raw data cleaning

The collected data is in a raw format. Some fields are encoded in hexadecimal according to the smart contract Application Binary Interface (ABI) [3]. To convert such fields into readable text we, indeed, need the ABI of the smart contract. The ABI is the interface of the smart contract. It is the de facto method for enconding Solidity contract calls for the EVM and, backwards. One of the encoded fields is *transaction.input* which represents the executed function name and the respective parameters.

Algorithm 2 Data decoding

1: $contractABI = getContractABI(contract)$
2: **for all** $transaction$ in $transactions$ **do**
3: $function, functionParams = decodeInput(contractABI, transaction.input)$
4: $transaction.function = function$
5: $transaction.functionParams = functionParams$
6: **end for**

In addition to the operation described above, fields not needed in the next steps are removed.

[1] `https://etherscan.io/`
[2] `https://ethtx.info/`
[3] `https://docs.soliditylang.org/en/v0.8.13/abi-spec.html`

The decoding step is not performed on internal transaction since the data obtained with EthTx contains both the raw data and the decoded data. Unnecessary fields are removed also there.

4.3 Sorting

As mentioned in Section 1, timestamps are not accurate because transactions mined in the same block have the same timestamp. To solve this issue, the user can specify a custom sorting criteria or use the transactionindex field.

4.4 Trace construction

As show in figure 4.1, the methodology can work with 3 different input data: transactions, transactions internal execution, and a combination of the two.

4.5 XES generation

5. Framework

5.1 Framework

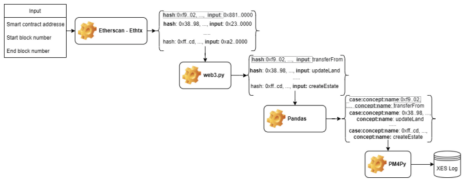

Figure 5.1: Framework

5.1.1 Etherscan

Etherscan is a blockchain explorer for the Ethereum network. It allows to search through transactions, blocks, wallet addresses, smart contracts, and other on-chain data.

It was used to collect the transactions data from the Ethereum blockchain about the specified smart contract addresses and within the selected block range.

5.1.2 Ethtx

EthTx Transaction Decoder[1] is an advanced decoder for blockchain transactions ("Tx") which translates complex transaction data into an organized and readable format that encompasses all triggered events, function calls, token movements etc.

Ethtx has been used to...

5.1.3 PM4Py

PM4Py[**PM4Py**] is a Python library that supports (state-of-the-art) Process Mining algorithms. It is completely open-source and intended to be used in both academia and industry projects. PM4Py is a product of the Fraunhofer Institute for Applied Information Technology. The main features are:

- Handling event data: importing and exporting event logs stored in various formats;
- Filtering event data: specific methods to filter an event log based on a timeframe, case performance, start/end activities, variants, and attribute values;

[1] `https://tokenflow.live/products/ethtx`

- Process discovery: provides alpha miner, inductive miner, heuristic miner, directly-follows graphs, and others...
- Petri Net management;
- Conformance checking: provides token-based replay and alignments
- Statistics: it is possible to calculate different statistics on top of classic event logs and data frames
- Log-Model evaluation: it is possible to compare the behavior contained in the log and the behavior contained in the model, to see if and how they match;
- Simulation: it offers different simulation algorithms, that starting from a model, can produce an output that follows the model and the different rules that have been provided by the user:

The library has been used to...

5.1.4 ProM

ProM is an Open Source framework for process mining algorithms. ProM provides a platform to users and developers of the process mining algorithms that is easy to use and easy to extend.

ProM has been used to...

6. Use case

6.1 Decentraland

6.2 Methodology application

6.3 Process Mining

6.4 Evaluation of results

For example, in our case study, we take the viewpoint of an individual kitty, but this may not be suitable for analysing the complete population. To generate different views, our framework allows analysts to materialize their choice of identity attributes in the manifest, and for some applications multiple manifests with different choices might be required to obtain the desired views. We mined the extracted event flows from both logs as depicted in Fig. 5. Fig. 5a shows that the developer started with two kitties initially, and bred them 3000 times for bootstrapping the game. The behavior during the everyday use (Fig. 5b) shows considerably more variation and includes all types of events. While we could delve into a deep analysis of kitty behaviour, the purpose of the case study in this paper was to test if the proposed framework can be applied to existing smart contracts – which we deem to be the case. We successfully extracted event logs, stored them in the XES format, and loaded them for analysis in both ProM and Disco.

7. Conclusions and Future Work

7.1 Conclusions

In this paper, we conducted a case study on process mining for data extracted from the blockchain application Augur. To this end, we used ELF to extract data over essentially the entire lifecycle of Augur v1.0. We used process mining methods and tools to explore the data, discover models for a set of variants, and conducted conformance checking and performance analyses. Finally, we interviewed the chief architect of Augur to validate our insights and understand their usefulness. As stated in Sect. 3, we followed open science principles and made all data and code from our study available publicly. In summary, we conclude that there is clear evidence for the usefulness of process mining on blockchain data. Main areas of interest for software developers may include user behavior analysis and security audits, for which we demonstrated the applicability of process mining tools. Indeed, we discovered a bug in Augur's smart contracts – albeit a non-critical one. Future research can be done 16 R. Hobeck, C. Klinkm¨uller, D. Bandara, I. Weber, W. van der Aalst evaluating other applications which might run on other blockchains, such as Hyperledger Fabric. The analysis method could be extended for blockchain-specific security and user studies, e.g., through drift detection and cohort analysis. [**apidoc**]

7.2 Future work

Bibliography

[1] Raffaele Conforti et al. "BPMN Miner: Automated discovery of BPMN process models with hierarchical structure". In: *Information Systems* 56 (2016), pp. 284–303. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2015.07.004. URL: https://www.sciencedirect.com/science/article/pii/S0306437915001325.

[2] Richard Hobeck et al. "Process Mining on Blockchain Data: A Case Study of Augur". In: Aug. 2021, pp. 306–323. ISBN: 978-3-030-85468-3. DOI: 10.1007/978-3-030-85469-0_20.

[3] IEEE. "IEEE standard for extensible event stream (XES) for achieving interoperability in event logs and event streams". In: *IEEE Std 1849-2016* (Nov 2016).

[4] Christopher Klinkmüller et al. "Mining Blockchain Processes: Extracting Process Mining Data from Blockchain Applications". In: Aug. 2019, pp. 71–86. ISBN: 978-3-030-30428-7. DOI: 10.1007/978-3-030-30429-4_6.

[5] Orlenys López-Pintado et al. "Caterpillar: A business process execution engine on the Ethereum blockchain". In: *Software: Practice and Experience* 49 (2019), pp. 1162 –1193.

[6] Roman Mühlberger et al. "Extracting Event Logs for Process Mining from Data Stored on the Blockchain". In: Sept. 2019.

[7] An Binh Tran, Qinghua Lu, and Ingo Weber. "Lorikeet: A Model-Driven Engineering Tool for Blockchain-Based Business Process Execution and Asset Management". In: *BPM*. 2018.

[8] Ingo Weber et al. "Untrusted Business Process Monitoring and Execution Using Blockchain". In: *BPM*. 2016.

Acknowledgments

Ringrazio...